1. Introduction to Database Administration

1.1. Types of Oracle Database Users

The types of users and their roles and responsibilities depend on the database site. A small site can have one database administrator who administers the database for application developers and users. A very large site can find it necessary to divide the duties of a database administrator among several people and among several areas of specialization.

**Database Administrators**

Each database requires at least one database administrator (DBA). An Oracle Database system can be large and can have many users. Therefore, database administration is sometimes not a one-person job, but a job for a group of DBAs who share responsibility. A database administrator's responsibilities can include the following tasks:

- Installing and upgrading the Oracle Database server and application tools.
- Allocating system storage and planning future storage requirements for the database system.
- Creating primary database storage structures (tablespaces) after application developers have designed an application.
- Creating primary objects (tables, views, indexes) once application developers have designed an application.
- Modifying the database structure, as necessary, from information given by application developers.
- Enrolling users and maintaining system security.
- Ensuring compliance with Oracle license agreements.
- Controlling and monitoring user access to the database.
- Monitoring and optimizing the performance of the database.
- Planning for backup and recovery of database information.
- Maintaining archived data on tape.
- Backing up and restoring the database.
- Contacting Oracle for technical support.

**Security Officers**

In some cases, a site assigns one or more security officers to a database. A security officer enrolls users, controls and monitors user access to the database, and maintains system security. As a DBA, you might not be responsible for these duties if your site has a separate security officer.

**Network Administrators**

Some sites have one or more network administrators. A network administrator, for example, administers Oracle networking products, such as Oracle Net Services.

**Database Users**

Database users interact with the database through applications or utilities. A typical user's responsibilities include the following tasks:

- Entering, modifying, and deleting data, where permitted
- Generating reports from the data

**Application Developers**

Application developers design and implement database applications. Their responsibilities include the following tasks:
• Designing and developing the database application
• Designing the database structure for an application
• Estimating storage requirements for an application
• Specifying modifications of the database structure for an application
• Relaying this information to a database administrator
• Tuning the application during development
• Establishing security measures for an application during development
• Application developers can perform some of these tasks in collaboration with DBAs.

**Application Administrators**

An Oracle Database site can assign one or more application administrators to administer a particular application. Each application can have its own administrator.

### 1.2. Tasks of a Database Administrator

The following tasks present a prioritized approach for designing, implementing, and maintaining an Oracle Database:

- **Task 1: Evaluate the Database Server Hardware**
- **Task 2: Install the Oracle Database Software**
- **Task 3: Plan the Database**
- **Task 4: Create and Open the Database**
- **Task 5: Back Up the Database**
- **Task 6: Enroll System Users**
- **Task 7: Implement the Database Design**
- **Task 8: Back Up the Fully Functional Database**
- **Task 9: Tune Database Performance**
- **Task 10: Download and Install Patches**
- **Task 11: Roll Out to Additional Hosts**

**Note:** When upgrading to a new release, back up your existing production environment, both software and database, before installation.

**Task 1: Evaluate the Database Server Hardware**
Evaluate how Oracle Database and its applications can best use the available computer resources. This evaluation should reveal the following information:

- How many disk drives are available to the Oracle Products?
- How many, if any, dedicated tape drives are available to Oracle products?
- How much memory is available to the instances of Oracle Database you will run?

**Task 2: Install the Oracle Database Software**
As the database administrator, you install the Oracle Database server software and any front-end tools and database applications that access the database. In some distributed processing installations, the database is controlled by a central computer (database server) and the database tools and applications are executed on remote computers (clients). In this case, you must also install the Oracle Net components necessary to connect the remote machines to the computer that executes Oracle Database.

**Task 3: Plan the Database**
As the database administrator, you must plan:

- The logical storage structure of the database
The overall database design
A backup strategy for the database

It is important to plan how the logical storage structure of the database will affect system performance and various database management operations. For example, before creating any tablespaces for your database, you should know how many datafiles will make up the tablespace, what type of information will be stored in each tablespace, and on which disk drives the datafiles will be physically stored. When planning the overall logical storage of the database structure, take into account the effects that this structure will have when the database is actually created and running. Consider how the logical storage structure of the database will affect:

- The performance of the computer executing running Oracle Database
- The performance of the database during data access operations
- The efficiency of backup and recovery procedures for the database

Plan the relational design of the database objects and the storage characteristics for each of these objects. By planning the relationship between each object and its physical storage before creating it, you can directly affect the performance of the database as a unit. Be sure to plan for the growth of the database. In distributed database environments, this planning stage is extremely important. The physical location of frequently accessed data dramatically affects application performance.

During the planning stage, develop a backup strategy for the database. You can alter the logical storage structure or design of the database to improve backup efficiency. It is beyond the scope of this book to discuss relational and distributed database design. If you are not familiar with such design issues, please refer to accepted industry-standard documentation.

Task 4: Create and Open the Database
After you complete the database design, you can create the database and open it for normal use. You can create a database at installation time, using the Database Configuration Assistant, or you can supply your own scripts for creating a database.

Task 5: Back Up the Database
After you create the database structure, carry out the backup strategy you planned for the database. Create any additional redo log files, take the first full database backup (online or offline), and schedule future database backups at regular intervals.

Task 6: Enroll System Users
After you back up the database structure, you can enroll the users of the database in accordance with your Oracle license agreement, and grant appropriate privileges and roles to these users.

Task 7: Implement the Database Design
After you create and start the database, and enroll the system users, you can implement the planned logical structure database by creating all necessary tablespaces. When you have finished creating tablespaces, you can create the database objects.

Task 8: Back Up the Fully Functional Database
When the database is fully implemented, again back up the database. In addition to regularly scheduled backups, you should always back up your database immediately after implementing changes to the database structure.

Task 9: Tune Database Performance
Optimizing the performance of the database is one of your ongoing responsibilities as a DBA. Oracle Database provides a database resource management feature that helps you to control the allocation of resources among various user groups.

Task 10: Download and Install Patches
After installation and on a regular basis, download and install patches. Patches are available as single interim
patches and as patchsets (or patch releases). Interim patches address individual software bugs and may or may not be needed at your installation. Patch releases are collections of bug fixes that are applicable for all customers. Patch releases have release numbers. For example, if you installed Oracle Database 10.2.0.0, the first patch release will have a release number of 10.2.0.1.

Task 11: Roll Out to Additional Hosts
After you have an Oracle Database installation properly configured, tuned, patched, and tested, you may want to roll that exact installation out to other hosts. Reasons to do this include the following:
You have multiple production database systems.
You want to create development and test systems that are identical to your production system.

Instead of installing, tuning, and patching on each additional host, you can clone your tested Oracle Database installation to other hosts, saving time and eliminating inconsistencies. There are two types of cloning available to you:

Cloning an Oracle home—just the configured and patched binaries from the Oracle home directory and subdirectories are copied to the destination host and fixed to match the new environment. You can then start an instance with this cloned home and creates a database. You can use the Enterprise Manager Clone Oracle Home tool to clone an Oracle home to one or more destination hosts. You can also manually clone an Oracle home using a set of provided scripts and Oracle Universal Installer.

Cloning a database—The tuned database, including database files, initialization parameters, and so on, are cloned to an existing Oracle home (possibly a cloned home). You can use the Enterprise Manager Clone Database tool to clone an Oracle database instance to an existing Oracle home.

Submitting Commands and SQL to the Database
The primary means of communicating with Oracle Database is by submitting SQL statements. Oracle Database also supports a superset of SQL, which includes commands for starting up and shutting down the database, modifying database configuration, and so on. There are three ways to submit these SQL statements and commands to Oracle Database:

Directly, using the command-line interface of SQL*Plus
Indirectly, using the graphical user interface of Oracle Enterprise Manager with Oracle Enterprise Manager (Enterprise Manager), you use an intuitive graphical interface to administer the database, and Enterprise Manager submits SQL statements and commands behind the scenes. Directly, using SQL Developer Developers use SQL Developer to create and test database schemas and applications, although you can also use it for database administration tasks.

1.3. About SQL*Plus
SQL*Plus is the primary command-line interface to your Oracle database. You use SQL*Plus to start up and shut down the database, set database initialization parameters, create and manage users, create and alter database objects (such as tables and indexes), insert and update data, run SQL queries, and more.

Before you can submit SQL statements and commands, you must connect to the database. With SQL*Plus, you can connect locally or remotely. Connecting locally means connecting to an Oracle database running on the same computer on which you are running SQL*Plus. Connecting remotely means connecting over a network to an Oracle database that is running on a remote computer. Such a database is referred to as a remote database. The SQL*Plus executable on the local computer is provided by a full Oracle Database installation, an Oracle Client installation, or an Instant Client installation.

1.3.1. Connecting to the Database with SQL*Plus
Oracle Database is composed of the Oracle instance, which is a collection of Oracle processes and memory, and a set of disk files that contain user data and system data. When you connect with SQL*Plus, you are connecting to the Oracle instance. Each instance has an instance ID, also known as a system ID (SID). Because there can be more than one Oracle instance on a host computer, each with its own set of data files, you must identify the instance to
which you want to connect. For a local connection, you identify the instance by setting operating system environment
variables. For a remote connection, you identify the instance by specifying a network address and a database service
name. In addition, for both local and remote connections, you must set environment variables to help the operating
system find the SQL*Plus executable and to provide the executable with a path to its support files and scripts.

To connect to an Oracle instance with SQL*Plus, therefore, you must complete the following steps:

Step 1: Open a Command Window
Step 2: Set Operating System Environment Variables
Step 3: Start SQL*Plus
Step 4: Submit the SQL*Plus CONNECT Statement

Step 1: Open a Command Window
Take the necessary action on your platform to open a window into which you can enter operating system
commands.

- Platform Action
  - UNIX and Linux Open a terminal session
  - Windows Open a Command Prompt window

Use this command window for steps 2 through 4.

Step 2: Set Operating System Environment Variables
Depending on your platform, you may have to set environment variables before starting SQL*Plus, or at least verify
that they are set properly. For example, on most platforms, ORACLE_SID and ORACLE_HOME must be set. In
addition, it is advisable to set the PATH environment variable to include the ORACLE_HOME/bin directory. Some
platforms may require additional environment variables. On the UNIX and Linux platforms, you must set environment
variables by entering operating system commands. On the Windows platform, Oracle Universal Installer (OUI)
automatically assigns values to ORACLE_HOME and ORACLE_SID in the Windows registry. If you did not create
a database upon installation, OUI does not set ORACLE_SID in the registry; after you create your database at a later
time, you must set the ORACLE_SID environment variable from a command window. UNIX and Linux installations
come with two scripts, oraenv and coraenv that you can use to easily set environment variables.

For all platforms, when switching between instances with different Oracle homes, you must change the
ORACLE_HOME environment variable. If multiple instances share the same Oracle home, you must change only
ORACLE_SID when switching instances.

Example: Setting Environment Variables in UNIX (C Shell)
setenv ORACLE_SID orcl
setenv ORACLE_HOME /u01/app/oracle/product/11.1.0/db_1
setenv LD_LIBRARY_PATH
$ORACLE_HOME/lib:/usr/lib:/usr/dt/lib:/usr/openwin/lib:/usr/ccs/lib

Example: Setting Environment Variables in Windows
SET ORACLE_SID=orcl

Example assumes that ORACLE_HOME is set in the registry and that ORACLE_SID is not set (or that you want to
override the registry value of ORACLE_SID to connect to a different instance). On Windows, environment variable
values that you set in a command prompt window override the values in the registry.

Step 3: Start SQL*Plus
To start SQL*Plus:
1. Do one of the following:
   Ensure that the PATH environment variable contains ORACLE_HOME/bin.
   Change directory to ORACLE_HOME/bin.
2. Enter the following command (case sensitive on UNIX and Linux):
   sqlplus /nolog

Step 4: Submit the SQL*Plus CONNECT Statement
You submit the SQL*Plus CONNECT statement to initially connect to the Oracle instance or at any time to reconnect as a different user. The syntax of the CONNECT statement is as follows:

```
CONNECT [logon] [AS {SYSOPER | SYSDBA}]
```

The syntax of logon is as follows:

```
{username | /}[@connect_identifier]
```

When you provide username, SQL*Plus prompts for a password. The password is not echoed as you type it. The following table describes the syntax components of the CONNECT statement.

<table>
<thead>
<tr>
<th>Syntax Component Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td>/</td>
</tr>
<tr>
<td>AS {SYSOPER</td>
</tr>
<tr>
<td>Username</td>
</tr>
<tr>
<td>connect_identifier (1)</td>
</tr>
</tbody>
</table>
| connect_identifier (2) | As an alternative, a connect identifier can use easy connect syntax. Easy connect provides out-of-the-box TCP/IP connectivity for remote databases without having to configure Oracle Net Services on the client (local) computer. Easy connect syntax for the connect identifier is as follows:

```
host[:port][/service_name]
```

where:
- `host` is the host name or IP address of the computer hosting the remote database.
- `port` is the TCP port on which the Oracle Net listener on host listens for database connections. If omitted, 1521 is assumed.
- `Service name` is the database service name. Can be omitted if the Net Services listener configuration on the remote host designates a default service. If no default service is configured, service_name must be supplied. Each database typically offers a service with a name equal to the global database name. |

Example:
This simple example connects to a local database as user SYSTEM. SQL*Plus prompts for the SYSTEM user password.

```
connect system
```

Example
This example connects to a local database as user SYS with the SYSDBA privilege. SQL*Plus prompts for the SYS user password.

```
connect sys as sysdba
```
When connecting as user SYS, you must connect AS SYSDBA.

Example
This example connects locally with operating system authentication.
connect /

Example
This example connects locally with the SYSDBA privilege with operating system authentication.
connect / as sysdba

Example
This example uses easy connect syntax to connect as user salesadmin to a remote database running on the host db1.mycompany.com. The Oracle Net listener (the listener) is listening on the default port (1521). The database service is sales.mycompany.com. SQL*Plus prompts for the salesadmin user password.connect salesadmin@db1.mycompany.com/sales.mycompany.com

Example
This example is identical that the listener is listening on the non-default port number 1522.
connect salesadmin@db1.mycompany.com:1522/sales.mycompany.com

Example
This example connects remotely as user salesadmin to the database service designated by the net service name sales1. SQL*Plus prompts for the salesadmin user password.
connect salesadmin@sales1

Example
This example connects remotely with external authentication to the database service designated by the net service name sales1.
connect /@sales1

Example
This example connects remotely with the SYSDBA privilege and with external authentication to the database service designated by the net service name sales1.
connect /@sales1 as sysdba

Because Oracle Database continues to evolve and can require maintenance, Oracle periodically produces new releases. Not all customers initially subscribe to a new release or require specific maintenance for their existing release. As a result, multiple releases of the product exist simultaneously. As many as five numbers may be required to fully identify a release. The significance of these numbers is discussed in the sections that follow.

1.4. Release Number Format

To understand the release nomenclature used by Oracle, examine the following example of an Oracle Database server labeled "Release 10.1.0.1.0".

Note: Starting with release 9.2, maintenance releases of Oracle Database are denoted by a change to the second digit of a release number. In previous releases, the third digit indicated a particular maintenance release.

**Major Database Release Number**
The first digit is the most general identifier. It represents a major new version of the software that contains significant new functionality.

**Database Maintenance Release Number**
The second digit represents a maintenance release level. Some new features may also be included.
Application Server Release Number
The third digit reflects the release level of the Oracle Application Server (OracleAS).

Component-Specific Release Number
The fourth digit identifies a release level specific to a component. Different components can have different numbers in
this position depending upon, for example, component patch sets or interim releases.

Platform-Specific Release Number
The fifth digit identifies a platform-specific release. Usually this is a patch set. When different platforms require the
equivalent patch set, this digit will be the same across the affected platforms.

1.4.1. Checking Your Current Release Number

- To identify the release of Oracle Database that is currently installed and to see the release levels of other
database components you are using, query the data dictionary view
PRODUCT_COMPONENT_VERSION. A sample query follows. (You can also query the V$VERSION
view to see component-level information.) Other product release levels may increment independent of the
database server.

```sql
COL PRODUCT FORMAT A35
COL VERSION FORMAT A15
COL STATUS FORMAT A15

SELECT * FROM PRODUCT_COMPONENT_VERSION;

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>VERSION</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLSRTL</td>
<td>10.2.0.1.0</td>
<td>Production</td>
</tr>
<tr>
<td>Oracle Database 10g Enterprise Edition</td>
<td>10.2.0.1.0</td>
<td>Prod</td>
</tr>
<tr>
<td>PL/SQL</td>
<td>10.2.0.1.0</td>
<td>Production</td>
</tr>
</tbody>
</table>

... It is important to convey to Oracle the results of this query when you report problems with the software.

1.5. About Database Administrator Security and Privileges

- To perform the administrative tasks of an Oracle Database DBA, you need specific privileges within the
database and possibly in the operating system of the server on which the database runs. Access to a
database administrator's account should be tightly controlled.

1.5.1. The Database Administrator's Operating System Account

To perform many of the administrative duties for a database, you must be able to execute operating system commands. Depending on the operating system on which Oracle Database is running, you might need an operating system account or ID to gain access to the operating system. If so, your operating system account might require operating system privileges or access rights that other database users do not require (for example, to perform Oracle Database software installation). Although you do not need the Oracle Database files to be stored in your account, you should have access to them.

1.5.2. Administrative User Accounts

Two administrative user accounts are automatically created when Oracle Database is installed:

- SYS (default password: CHANGE_ON_INSTALL)
- SYSTEM (default password: MANAGER)

Note: Both Oracle Universal Installer (OUI) and Database Configuration Assistant (DBCA) now prompt for SYS and
SYSTEM passwords and do not accept the default passwords “change_on_install” or “manager”, respectively. If you create the database manually, Oracle strongly recommends that you specify passwords for SYS and SYSTEM at database creation time, rather than using these default passwords.

- Create at least one additional administrative user and grant to that user an appropriate administrative role to use when performing daily administrative tasks. Do not use SYS and SYSTEM for these purposes.
- Note Regarding Security Enhancements: In this release of Oracle Database and in subsequent releases, several enhancements are being made to ensure the security of default database user accounts.

SYS
When you create an Oracle Database, the user SYS is automatically created and granted the DBA role. All of the base tables and views for the database data dictionary are stored in the schema SYS. These base tables and views are critical for the operation of Oracle Database. To maintain the integrity of the data dictionary, tables in the SYS schema are manipulated only by the database. They should never be modified by any user or database administrator, and no one should create any tables in the schema of user SYS. (However, you can change the storage parameters of the data dictionary settings if necessary.) Ensure that most database users are never able to connect to Oracle Database using the SYS account.

SYSTEM
When you create an Oracle Database, the user SYSTEM is also automatically created and granted the DBA role. The SYSTEM username is used to create additional tables and views that display administrative information, and internal tables and views used by various Oracle Database options and tools. Never use the SYSTEM schema to store tables of interest to non-administrative users.

1.5.3. The DBA Role
A predefined DBA role is automatically created with every Oracle Database installation. This role contains most database system privileges. Therefore, the DBA role should be granted only to actual database administrators.

Note: The DBA role does not include the SYSDBA or SYSOPER system privileges. These are special administrative privileges that allow an administrator to perform basic database administration tasks, such as creating the database and instance startup and shutdown.

1.5.4. Database Administrator Authentication
As a DBA, you often perform special operations such as shutting down or starting up a database. Because only a DBA should perform these operations, the database administrator usernames require a secure authentication scheme.

Administrative Privileges
Administrative privileges that are required for an administrator to perform basic database operations are granted through two special system privileges, SYSDBA and SYSOPER. You must have one of these privileges granted to you, depending upon the level of authorization you require.

Note: The SYSDBA and SYSOPER system privileges allow access to a database instance even when the database is not open. Control of these privileges is totally outside of the database itself.

The SYSDBA and SYSOPER privileges can also be thought of as types of connections that enable you to perform certain database operations for which privileges cannot be granted in any other fashion. For example, you if you have the SYSDBA privilege, you can connect to the database by specifying CONNECT AS SYSDBA.

SYSDBA and SYSOPER
The following operations are authorized by the SYSDBA and SYSOPER system privileges:

<table>
<thead>
<tr>
<th>System Privilege</th>
<th>Operations Authorized</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDBA</td>
<td>Perform STARTUP and SHUTDOWN operations</td>
</tr>
<tr>
<td></td>
<td>ALTER DATABASE: open, mount, back up, or change character set</td>
</tr>
</tbody>
</table>
The manner in which you are authorized to use these privileges depends upon the method of authentication that you use. When you connect with SYSDBA or SYSOPER privileges, you connect with a default schema, not with the schema that is generally associated with your username. For SYSDBA this schema is SYS; for SYSOPER the schema is PUBLIC.

Connecting with Administrative Privileges: Example
This example illustrates that a user is assigned another schema (SYS) when connecting with the SYSDBA system privilege. Assume that the sample user oe has been granted the SYSDBA system privilege and has issued the following statements:

```
CONNECT oe
CREATE TABLE admin_test (name VARCHAR2(20));
```

Later, user oe issues these statements:

```
CONNECT oe AS SYSDBA
SELECT * FROM admin_test;
```

User oe now receives the following error:

```
ORA-00942: table or view does not exist
```

Having connected as SYSDBA, user oe now references the SYS schema, but the table was created in the oe schema.

### 1.5.5. Authentication Method for Database Administrators

Database Administrators can authenticate through the database data dictionary, (using an account password) like other users. Keep in mind that beginning with Oracle Database 11g Release 1, database passwords are case sensitive. (You can disable case sensitivity and return to pre–Release 11g behavior by setting the SEC_CASE_SENSITIVE_LOGON initialization parameter to FALSE.). In addition to normal data dictionary authentication, the following methods are available for authenticating database administrators with the SYSDBA or SYSOPER privilege:

- Operating system (OS) authentication
- A password file
- Strong authentication with a network-based authentication service, such as Oracle Internet Directory

These methods are required to authenticate a database administrator when the database is not started or otherwise unavailable. (They can also be used when the database is available.) The remainder of this section focuses on operating system authentication and password file authentication.
Notes: These methods replace the CONNECT INTERNAL syntax provided with earlier versions of Oracle Database. CONNECT INTERNAL is no longer supported.

Operating system authentication takes precedence over password file authentication. If you meet the requirements for operating system authentication, then even if you use a password file, you will be authenticated by operating system authentication. Your choice will be influenced by whether you intend to administer your database locally on the same machine where the database resides, or whether you intend to administer many different databases from a single remote client. Figure illustrates the choices you have for database administrator authentication schemes.

If you are performing remote database administration, consult your Oracle Net documentation to determine whether you are using a secure connection. Most popular connection protocols, such as TCP/IP and DECnet, are not secure.

Non-secure Remote Connections
To connect to Oracle Database as a privileged user over a nonsecure connection, you must be authenticated by a password file. When using password file authentication, the database uses a password file to keep track of database usernames that have been granted the SYSDBA or SYSOPER system privilege.

Local Connections and Secure Remote Connections
You can connect to Oracle Database as a privileged user over a local connection or a secure remote connection in a way:

- If the database has a password file and you have been granted the SYSDBA or SYSOPER system privilege, then you can connect and be authenticated by a password file.
- If the server is not using a password file, or if you have not been granted SYSDBA or SYSOPER privileges and are therefore not in the password file, you can use operating system authentication. On most operating systems, authentication for database administrators involves placing the operating system username of the database administrator in a special group, generically referred to as OSDBA. Users in that group are granted SYSDBA privileges. A similar group, OSOPER, is used to grant SYSOPER privileges to users.
1.5.6. Using Operating System Authentication

This section describes how to authenticate an administrator using the operating system.

OSDBA and OSOPER

Two special operating system groups control database administrator connections when using operating system authentication. These groups are generically referred to as OSDBA and OSOPER. The groups are created and assigned specific names as part of the database installation process. The specific names vary depending upon your operating system and are listed in the following table:

<table>
<thead>
<tr>
<th>Operating System Group</th>
<th>UNIX User Group</th>
<th>Windows User Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSDBA</td>
<td>dba</td>
<td></td>
</tr>
<tr>
<td>OSOPER</td>
<td>oper</td>
<td>ORA_OPER</td>
</tr>
<tr>
<td>ORA_DBA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The default names assumed by the Oracle Universal Installer can be overridden. How you create the OSDBA and OSOPER groups is operating system specific. Membership in the OSDBA or OSOPER group affects your connection to the database in the following ways:

- If you are a member of the OSDBA group and you specify AS SYSDBA when you connect to the database, then you connect to the database with the SYSDBA system privilege.
- If you are a member of the OSOPER group and you specify AS SYSOPER when you connect to the database, then you connect to the database with the SYSOPER system privilege. If you are not a member of either of these operating system groups and you attempt to connect as SYSDBA or SYSOPER, the CONNECT command fails.

1.5.7. Preparing to Use Operating System Authentication

To enable operating system authentication of an administrative user:
1. Create an operating system account for the user.
2. Add the account to the OSDBA or OSOPER operating system defined groups.

Connecting Using Operating System Authentication

A user can be authenticated, enabled as an administrative user, and connected to a local database by typing one of the following SQL*Plus commands:

```
CONNECT / AS SYSDBA
CONNECT / AS SYSOPER
```

For the Windows platform only, remote operating system authentication over a secure connection is supported. You must specify the net service name for the remote database:

```
CONNECT /@net_service_name AS SYSDBA
CONNECT /@net_service_name AS SYSOPER
```

Both the client computer and database host computer must be on a Windows domain.

Using Password File Authentication

This section describes how to authenticate an administrative user using password file authentication.

Preparing to Use Password File Authentication

To enable authentication of an administrative user using password file authentication you must do the following:
1. If not already created, create the password file using the ORAPWD utility:

```
ORAPWD FILE=filename ENTRIES=max_users
```

Note:
• When you invoke Database Configuration Assistant (DBCA) as part of the Oracle Database installation process, DBCA creates a password file.
• Beginning with Oracle Database 11g Release 1, passwords in the password file are case sensitive unless you include the `IGNORECASE=Y` command-line argument.

2. Set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter to `EXCLUSIVE`. (This is the default).

**Note:** `REMOTE_LOGIN_PASSWORDFILE` is a static initialization parameter and therefore cannot be changed without restarting the database.

3. Connect to the database as user SYS (or as another user with the administrative privileges).
4. If the user does not already exist in the database, create the user and assign a password.
   Keep in mind that beginning with Oracle Database 11g Release 1, database passwords are case sensitive. (You can disable case sensitivity and return to pre–Release 11g behavior by setting the `SEC_CASE_SENSITIVE_LOGON` initialization parameter to `FALSE`.
5. Grant the SYSDBA or SYSOPER system privilege to the user:

   ```sql
   GRANT SYSDBA to oe;
   ```

   This statement adds the user to the password file, thereby enabling connection AS SYSDBA.

### 1.5.8. Connecting Using Password File Authentication

Administrative users can be connected and authenticated to a local or remote database by using the SQL*Plus `CONNECT` command. They must connect using their username and password and the AS SYSDBA or AS SYSOPER clause. Note that beginning with Oracle Database 11g Release 1, passwords are case-sensitive unless the password file was created with the `IGNORECASE=Y` option.

For example, user oe has been granted the SYSDBA privilege, so oe can connect as follows:

```
CONNECT oe AS SYSDBA
```

However, user oe has not been granted the SYSOPER privilege, so the following command will fail:

```
CONNECT oe AS SYSOPER
```

**Note:** Operating system authentication takes precedence over password file authentication. Specifically, if you are a member of the OSDBA or OSOPER group for the operating system, and you connect as SYSDBA or SYSOPER, you will be connected with associated administrative privileges regardless of the username/password that you specify. If you are not in the OSDBA or OSOPER groups, and you are not in the password file, then attempting to connect as SYSDBA or as SYSOPER fails.

### 1.5.9. Creating and Maintaining a Password File

You can create a password file using the password file creation utility, ORAPWD. For some operating systems, you can create this file as part of your standard installation.

#### Using ORAPWD

The syntax of the ORAPWD command is as follows:

```sql
ORAPWD FILE=filename [ENTRIES=numusers] [FORCE={Y|N}] [IGNORECASE={Y|N}] [NOSYSDBA={Y|N}]
```

Command arguments are summarized in the following table.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Name to assign to the password file. You must supply a complete path. If you supply only a file name, the file is written to the current directory.</td>
</tr>
<tr>
<td>ENTRIES</td>
<td>(Optional) Maximum number of entries (user accounts) to permit in the file.</td>
</tr>
</tbody>
</table>
FORCE (Optional) If y, permits overwriting an existing password file.

IGNORECASE (Optional) If y, passwords are treated as case-insensitive. NOSYSDBA (Optional) For Data Vault installations.

<table>
<thead>
<tr>
<th>FORCE</th>
<th>(Optional) If y, permits overwriting an existing password file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGNORECASE</td>
<td>(Optional) If y, passwords are treated as case-insensitive.</td>
</tr>
</tbody>
</table>

There are no spaces permitted around the equal-to (=) character.
The command prompts for the SYS password and stores the password in the created password file.

Example
The following command creates a password file named orapworcl that allows up to 30 privileged users with different passwords.

```
orapwd FILE=orapworcl ENTRIES=30
```

**ORAPWD Command Line Argument Descriptions**
The following sections describe the ORAPWD command line arguments.

**FILE**
This argument sets the name of the password file being created. You must specify the full path name for the file. If you supply only a file name, the file is written to the current directory. The contents of this file are encrypted, and the file cannot be read directly. This argument is mandatory. The types of filenames allowed for the password file are operating system specific.

Some operating systems require the password file to adhere to a specific format and be located in a specific directory. Other operating systems allow the use of environment variables to specify the name and location of the password file. If you are running multiple instances of Oracle Database using Oracle Real Application Clusters, the environment variable for each instance should point to the same password file.

**Caution:** It is critically important to the security of your system that you protect your password file and the environment variables that identify the location of the password file. Any user with access to these could potentially compromise the security of the connection.

**ENTRIES**
This argument specifies the number of entries that you require the password file to accept. This number corresponds to the number of distinct users allowed to connect to the database as SYSDBA or SYSOPER. The actual number of allowable entries can be higher than the number of users, because the ORAPWD utility continues to assign password entries until an operating system block is filled. For example, if your operating system block size is 512 bytes, it holds four password entries. The number of password entries allocated is always a multiple of four. Entries can be reused as users are added to and removed from the password file. If you intend to specify REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE, and to allow the granting of SYSDBA and SYSOPER privileges to users, this argument is required.

**Caution:** When you exceed the allocated number of password entries, you must create a new password file. To avoid this necessity, allocate a number of entries that is larger than you think you will ever need.

**FORCE**
This argument, if set to Y, enables you to overwrite an existing password file. An error is returned if a password file of the same name already exists and this argument is omitted or set to N.

**IGNORECASE**
If this argument is set to y, passwords are case-insensitive. That is, case is ignored when comparing the password that the user supplies during login with the password in the password file.

**Setting REMOTE_LOGIN_PASSWORDFILE**
In addition to creating the password file, you must also set the initialization parameter REMOTE_LOGIN_PASSWORDFILE to the appropriate value. The values recognized are:
NONE: Setting this parameter to NONE causes Oracle Database to behave as if the password file does not exist. That is, no privileged connections are allowed over non-secure connections.

EXCLUSIVE: (The default) An EXCLUSIVE password file can be used with only one instance of one database. Only an EXCLUSIVE file can be modified. Using an EXCLUSIVE password file enables you to add, modify, and delete users. It also enables you to change the SYS password with the ALTER USER command.

SHARED: A SHARED password file can be used by multiple databases running on the same server, or multiple instances of an Oracle Real Application Clusters (RAC) database. A SHARED password file cannot be modified. This means that you cannot add users to a SHARED password file. Any attempt to do so or to change the password of SYS or other users with the SYSDBA or SYSOPER privileges generates an error. All users needing SYSDBA or SYSOPER system privileges must be added to the password file when REMOTE_LOGIN_PASSWORDFILE is set to EXCLUSIVE. After all users are added, you can change REMOTE_LOGIN_PASSWORDFILE to SHARED, and then share the file. This option is useful if you are administering multiple databases or a RAC database. If REMOTE_LOGIN_PASSWORDFILE is set to EXCLUSIVE or SHARED and the password file is missing, this is equivalent to setting REMOTE_LOGIN_PASSWORDFILE to NONE.

Note: You cannot change the password for SYS if REMOTE_LOGIN_PASSWORDFILE is set to SHARED. An error message is issued if you attempt to do so.

1.5.10. Adding Users to a Password File

When you grant SYSDBA or SYSOPER privileges to a user, that user's name and privilege information are added to the password file. If the server does not have an EXCLUSIVE password file (that is, if the initialization parameter REMOTE_LOGIN_PASSWORDFILE is NONE or SHARED, or the password file is missing), Oracle Database issues an error if you attempt to grant these privileges. A user's name remains in the password file only as long as that user has at least one of these two privileges. If you revoke both of these privileges, Oracle Database removes the user from the password file.

Creating a Password File and Adding New Users to It

Use the following procedure to create a password and add new users to it:
1. Follow the instructions for creating a password file as explained in "Using ORAPWD"
2. Set the REMOTE_LOGIN_PASSWORDFILE initialization parameter to EXCLUSIVE.

Note: REMOTE_LOGIN_PASSWORDFILE is a static initialization parameter and therefore cannot be changed without restarting the database.

3. Connect with SYSDBA privileges as shown in the following example, and enter the SYS password when prompted:

   CONNECT SYS AS SYSDBA

4. Start up the instance and create the database if necessary, or mount and open an existing database.
5. Create users as necessary. Grant SYSDBA or SYSOPER privileges to yourself and other users as appropriate.

Granting and Revoking SYSDBA and SYSOPER Privileges

If your server is using an EXCLUSIVE password file, use the GRANT statement to grant the SYSDBA or SYSOPER system privilege to a user, as shown in the following example:

   GRANT SYSDBA TO oe;

Use the REVOKE statement to revoke the SYSDBA or SYSOPER system privilege from a user, as shown in the following example:

   REVOKE SYSDBA FROM oe;

Because SYSDBA and SYSOPER are the most powerful database privileges, the WITH ADMIN OPTION is not used in the GRANT statement. That is, the grantee cannot in turn grant the SYSDBA or SYSOPER privilege to another user. Only a user currently connected as SYSDBA can grant or revoke another user's SYSDBA or
SYSOPER system privileges. These privileges cannot be granted to roles, because roles are available only after database startup. Do not confuse the SYSDBA and SYSOPER database privileges with operating system roles.

**Viewing Password File Members**
Use the V$PWFILE_USERS view to see the users who have been granted SYSDBA or SYSOPER system privileges for a database. The columns displayed by this view are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERNAME</td>
<td>This column contains the name of the user that is recognized by the password file.</td>
</tr>
<tr>
<td>SYSDBA</td>
<td>If the value of this column is TRUE, then the user can log on with SYSDBA system privileges.</td>
</tr>
<tr>
<td>SYSOPER</td>
<td>If the value of this column is TRUE, then the user can log on with SYSOPER system privileges.</td>
</tr>
</tbody>
</table>

**1.5.11. Maintaining a Password File**

**Expanding the Number of Password File Users**
If you receive the file full error (ORA-1996) when you try to grant SYSDBA or SYSOPER system privileges to a user, you must create a larger password file and regrant the privileges to the users.

**Replacing a Password File**
Use the following procedure to replace a password file:
1. Identify the users who have SYSDBA or SYSOPER privileges by querying the V$PWFILE_USERS view.
2. Delete the existing password file.
3. Follow the instructions for creating a new password file using the ORAPWD utility in "Using ORAPWD". Ensure that the ENTRIES parameter is set to a number larger than you think you will ever need.

**Removing a Password File**
If you determine that you no longer require a password file to authenticate users, you can delete the password file and then optionally reset the REMOTE_LOGIN_PASSWORDFILE initialization parameter to NONE. After you remove this file, only those users who can be authenticated by the operating system can perform SYSDBA or SYSOPER database administration operations.

**1.6. Data Utilities**
Oracle utilities are available to help you maintain the data in your Oracle Database.

**SQL*Loader**
SQL*Loader is used both by database administrators and by other users of Oracle Database. It loads data from standard operating system files (such as, files in text or C data format) into database tables.

**Export and Import Utilities**
The Data Pump utility enables you to archive data and to move existing data between one Oracle Database and another. Also available is the original Import utility for importing data from earlier releases. Beginning with Release 11g, the original Export utility is desupported for general use and is recommended only in very specific circumstances.
2. Oracle Database Creation and Configuration

2.1. About Creating an Oracle Database

After you plan your database using some of the guidelines presented in this section, you can create the database with a graphical tool or a SQL command. You typically create a database during Oracle Database software installation. However, you can also create a database after installation. Reasons to create a database after installation are as follows:

- You used Oracle Universal Installer (OUI) to install software only, and did not create a database.
- You want to create another database (and database instance) on the same host computer as an existing Oracle database. In this case, this assumes that the new database uses the same Oracle home as the existing database. You can also create the database in a new Oracle home by running OUI again.
- You want to make a copy of (clone) a database. The specific methods for creating a database are:
  - With Database Configuration Assistant (DBCA), a graphical tool.
  - With the CREATE DATABASE SQL statement.

2.2. Considerations before Creating the Database

Database creation prepares several operating system files to work together as an Oracle Database. You need only create a database once, regardless of how many datafiles it has or how many instances access it. You can create a database to erase information in an existing database and create a new database with the same name and physical structure.

2.2.1. Meeting Creation Prerequisites

Before you can create a new database, the following prerequisites must be met:

- The desired Oracle software must be installed. This includes setting various environment variables unique to your operating system and establishing the directory structure for software and database files.
- Sufficient memory must be available to start the Oracle Database instance.
- Sufficient disk storage space must be available for the planned database on the computer that runs Oracle Database.

2.2.2. Creating a Database with DBCA

Database Configuration Assistant (DBCA) provides a graphical interface and guided workflow for creating and configuring a database. It is the preferred way to create a database, because it is a more automated approach, and your database is ready to use when DBCA completes. DBCA can be launched by the Oracle Universal Installer (OUI), depending upon the type of install that you select. You can also launch DBCA as a standalone tool at any time after Oracle Database installation. The following example creates a database by passing command-line arguments to DBCA:

```
$ dbca -silent -createDatabase -templateName General_Purpose.dbc -dbname oracle11g -sid oracle11g -responseFile NO_VALUE -characterSet AL32UTF8 -memoryPercentage 30 -emConfiguration LOCAL
Enter SYSTEM user password: password
Enter SYS user password: password
Copying database files
1% complete
```
To ensure completely silent operation, you can redirect stdout to a file. If you do this, however, you must supply passwords for the administrative accounts in command-line arguments or the response file.

To view brief help for DBCA command-line arguments, enter the following command:

dbca -help

Creating a Database with the CREATE DATABASE Statement

Using the CREATE DATABASE SQL statement is a more manual approach to creating a database. One advantage of using the CREATE DATABASE statement is that you can use it in a script and run the script as often as necessary. If you use the CREATE DATABASE statement, you must complete additional actions before you have an operational database. These actions include building views on the data dictionary tables and installing standard PL/SQL packages. You perform these actions by running prepared scripts. If you have existing scripts for creating your database, consider editing those scripts to take advantage of new Oracle Database features. The instructions in this section apply to single-instance installations only.

Note: Single-instance does not mean that only one Oracle instance can reside on a single host computer. In fact, multiple Oracle instances (and their associated databases) can run on a single host computer. A single-instance database is a database that is accessed by only one Oracle instance, as opposed to an Oracle RAC database, which is accessed concurrently by multiple Oracle instances on multiple nodes.

Complete the following steps to create a database with the CREATE DATABASE statement. The examples create a database named mynewdb.

Tip: If you are using Oracle Automatic Storage Management (ASM) to manage your disk storage, you must start the ASM instance and configure your disk groups before performing these steps.

Step 1: Specify an Instance Identifier (SID)

Decide on a unique Oracle system identifier (SID) for your instance, open a command window, and set the ORACLE_SID environment variable. ORACLE_SID is used to distinguish this instance from other Oracle Database instances that you may create later and run concurrently on the same host computer. The maximum number of characters for ORACLE_SID is 12, and only letters and numeric digits are permitted. On some platforms, the SID is case-sensitive.

Note: It is common practice to set the SID to be equal to the database name. The maximum number of characters for the database name is eight.

The following example for UNIX and Linux operating systems sets the SID for the instance that you will connect to in Step 6: Connect to the Instance:

Bourne, Bash, or Korn shell:

```bash
ORACLE_SID=mynewdb
export ORACLE_SID
```

C shell:

```bash
setenv ORACLE_SID mynewdb
```

The following example sets the SID for the Windows operating system:

```bash
set ORACLE_SID=mynewdb
```

Step 2: Ensure That the Required Environment Variables Are Set

Depending on your platform, before you can start SQL*Plus (as required in Step 6: Connect to the Instance), you may have to set environment variables, or at least verify that they are set properly. For example, on most platforms, ORACLE_SID and ORACLE_HOME must be set. In addition, it is advisable to set the PATH variable to include the ORACLE_HOME/bin directory. On the UNIX and Linux platforms, you must set these environment variables manually. On the Windows platform, OUI automatically assigns values to ORACLE_HOME and ORACLE_SID in the Windows registry. If you did not create a database upon installation, OUI does not set ORACLE_SID in the
Step 3: Choose a Database Administrator Authentication Method

You must be authenticated and granted appropriate system privileges in order to create a database. You can authenticate as an administrator with the required privileges in the following ways:

- With a password file
- With operating system authentication

In this step, you decide on an authentication method. If you decide to authenticate with a password file, create the password file. If you decide to authenticate with operating system authentication, ensure that you log in to the host computer with a user account that is a member of the appropriate operating system user group. On the UNIX and Linux platforms, for example, this is typically the dba user group. On the Windows platform, the user installing the Oracle software is automatically placed in the required user group.

Step 4: Create the Initialization Parameter File

When an Oracle instance starts, it reads an initialization parameter file. This file can be a text file, which can be created and modified with a text editor, or a binary file, which is created and dynamically modified by the database. The binary file, which is preferred, is called a server parameter file. In this step, you create a text initialization parameter file. In a later step, you create a server parameter file from the text file.

For convenience, store your initialization parameter file in the Oracle Database default location, using the default file name. Then when you start your database, it will not be necessary to specify the PFILE clause of the STARTUP command, because Oracle Database automatically looks in the default location for the initialization parameter file.

Step 5: (Windows Only) Create an Instance

On the Windows platform, before you can connect to an instance, you must manually create it if it does not already exist. The ORADIM command creates an Oracle instance by creating a new Windows service.

To create an instance:

Enter the following command at a Windows command prompt:

```
$oradim -NEW -SID sid -STARTMODE MANUAL -PFILE pfile
```

Where sid is the desired SID (for example mynewdb) and pfile is the full path to the text initialization parameter file. This command creates the instance but does not start it.

Warning: Do not set the -STARTMODE argument to AUTO at this point, because this causes the new instance to start and attempt to mount the database, which does not exist yet. You can change this parameter to AUTO.

Step 6: Connect to the Instance

Start SQL*Plus and connect to your Oracle Database instance with the SYSDBA system privilege.

To authenticate with a password file, enter the following commands, and then enter the SYS password when prompted:

```
$ sqlplus /nolog
SQL> CONNECT SYS AS SYSDBA
```

To authenticate with operating system authentication, enter the following commands:

```
$ sqlplus /nolog
SQL> CONNECT / AS SYSDBA
```

SQL*Plus outputs the following message:

```
Connected to an idle instance.
```

**Note:** SQL*Plus may output a message similar to the following:

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
```

registry, and you will have to set the ORACLE_SID environment variable when you create your database later.
With the Partitioning, OLAP and Data Mining options
If so, this means that the instance is already started. You may have connected to the wrong instance. Exit SQL*Plus with the EXIT command, check that ORACLE_SID is set properly, and repeat this step.

Step 7: Create a Server Parameter File
The server parameter file enables you to change initialization parameters with the ALTER SYSTEM command and persist the changes across a database shutdown and startup. You create the server parameter file from your edited text initialization file. The following SQL*Plus command reads the text initialization parameter file (PFILE) with the default name from the default location, creates a server parameter file (SPFILE) from the text initialization parameter file, and writes the SPFILE to the default location with the default SPFILE name.

`CREATE SPFILE FROM PFILE;`

You can also supply the file name and path for both the PFILE and SPFILE if you are not using default names and locations.

**Tip:** The database must be restarted before the server parameter file takes effect.

**Note:** Although creating a server parameter file is optional at this point, it is recommended. If you do not create a server parameter file, the instance continues to read the text initialization parameter file whenever it starts.

Important—If you are using Oracle-managed files and your initialization parameter file does not contain the CONTROL_FILES parameter, you must create a server parameter file now so the database can save the names and location of the control files that it creates during the CREATE DATABASE statement.

Step 8: Start the Instance
Start an instance without mounting a database. Typically, you do this only during database creation or while performing maintenance on the database. Use the STARTUP command with the NOMOUNT clause. In this example, because the initialization parameter file or server parameter file is stored in the default location, you are not required to specify the PFILE clause:

`STARTUP NOMOUNT`

At this point, the instance memory is allocated and its processes are started. The database itself does not yet exist.

Step 9: Issue the CREATE DATABASE Statement
To create the new database, use the CREATE DATABASE statement.

Example
The following statement creates database mynewdb. This database name must agree with the DB_NAME parameter in the initialization parameter file. This example assumes the following:

The initialization parameter file specifies the number and location of control files with the CONTROL_FILES parameter.

The directory /u01/app/oracle/oradata/mynewdb exists.

`CREATE DATABASE mynewdb`

```
USER SYS IDENTIFIED BY sys_password
USER SYSTEM IDENTIFIED BY system_password
LOGFILE GROUP 1 ('/u01/app/oracle/oradata/mynewdb/redo01.log') SIZE 100M,
   GROUP 2 ('/u01/app/oracle/oradata/mynewdb/redo02.log') SIZE 100M,
   GROUP 3 ('/u01/app/oracle/oradata/mynewdb/redo03.log') SIZE 100M
MAXLOGFILES 5
MAXLOGMEMBERS 5
MAXLOGHISTORY 1
MAXDATAFILES 100
CHARACTER SET US7ASCII
NATIONAL CHARACTER SET AL16UTF16
EXTENT MANAGEMENT LOCAL
```
A database is created with the following characteristics:

- The database is named mynewdb. Its global database name is mynewdb.us.oracle.com, where the domain portion (us.oracle.com) is taken from the initialization file.
- Three control files are created as specified by the CONTROL_FILES initialization parameter, which was set before database creation in the initialization parameter file.

The passwords for user accounts SYS and SYSTEM are set to the values that you specified. Beginning with Release 11g, the passwords are case-sensitive. The two clauses that specify the passwords for SYS and SYSTEM are not mandatory in this release of Oracle Database. However, if you specify either clause, you must specify both clauses. The new database has three redo log files as specified in the LOGFILE clause. MAXLOGFILES, MAXLOGMEMBERS, and MAXLOGHISTORY define limits for the redo log. MAXDATAFILES specifies the maximum number of datafiles that can be open in the database. This number affects the initial sizing of the control file.

**Note:** You can set several limits during database creation. Some of these limits are limited by and affected by operating system limits. For example, if you set MAXDATAFILES, Oracle Database allocates enough space in the control file to store MAXDATAFILES filenames, even if the database has only one datafile initially. However, because the maximum control file size is limited and operating system dependent, you might not be able to set all CREATE DATABASE parameters at their theoretical maximums.

- The US7ASCII character set is used to store data in this database.
- The AL16UTF16 character set is specified as the NATIONAL CHARACTER SET, used to store data in columns specifically defined as NCHAR, NCLOB, or NVARCHAR2.
- The SYSTEM tablespace, consisting of the operating system file /u01/app/oracle/oradata/mynewdb/system01.dbf is created as specified by the DATAFILE clause. If a file with that name already exists, it is overwritten.
- The SYSTEM tablespace is created as a locally managed tablespace.
- A SYSAUX tablespace is created, consisting of the operating system file /u01/app/oracle/oradata/mynewdb/syaux01.dbf as specified in the SYSAUX DATAFILE clause.
- The DEFAULT TABLESPACE clause creates and names a default permanent tablespace for this database.
- The DEFAULT TEMPORARY TABLESPACE clause creates and names a default temporary tablespace for this database.
- The UNDO TABLESPACE clause creates and names an undo tablespace that is used to store undo data for this database if you have specified
- UNDO_MANAGEMENT=AUTO in the initialization parameter file. If you omit this parameter, it defaults to AUTO.
Redo log files will not initially be archived, because the ARCHIVELOG clause is not specified in this CREATE DATABASE statement. This is customary during database creation. You can later use an ALTER DATABASE statement to switch to ARCHIVELOG mode. The initialization parameters in the initialization parameter file for mynewdb relating to archiving are LOG_ARCHIVE_DEST_1 and LOG_ARCHIVE_FORMAT.

Tips: Ensure that all directories exist. The CREATE DATABASE statement does not create directories. If you are not using Oracle-managed files, every tablespace clause must include a DATAFILE or TEMPFILE clause. If you receive an error message that contains a process number, examine the trace file for that process. Look for the trace file that contains the process number in the trace file name. Use the alert log as another source for investigating errors.

Example
This example illustrates creating a database with Oracle Managed Files, which enables you to use a much simpler CREATE DATABASE statement. To use Oracle Managed Files, the initialization parameter DB_CREATE_FILE_DEST must be set. This parameter defines the base directory for the various database files that the database creates and automatically names. The following statement is an example of setting this parameter in the initialization parameter file:

```
DB_CREATE_FILE_DEST='/u01/app/oracle/oradata'
```

With Oracle Managed Files and the following CREATE DATABASE statement, the database creates the SYSTEM and SYSAUX tablespaces, creates the additional tablespaces specified in the statement, and chooses default sizes and properties for all datafiles, control files, and redo log files. Note that these properties and the other default database properties set by this method may not be suitable for your production environment, so it is recommended that you examine the resulting configuration and modify it if necessary.

```
CREATE DATABASE mynewdb
USER SYS IDENTIFIED BY sys_password
USER SYSTEM IDENTIFIED BY system_password
EXTENT MANAGEMENT LOCAL
DEFAULT TEMPORARY TABLESPACE temp
UNDO TABLESPACE undotbs1
DEFAULT TABLESPACE users;
```

Tips: If your CREATE DATABASE statement fails, and if you did not complete Step 7, ensure that there is not a pre-existing server parameter file (SPFILE) for this instance that is setting initialization parameters in an unexpected way. For example, an SPFILE contains a setting for the complete path to all control files, and the CREATE DATABASE statement fails if those control files do not exist. Ensure that you shut down and restart the instance (with STARTUP NOMOUNT) after removing an unwanted SPFILE.

Step 10: Create Additional Tablespaces
To make the database functional, you need to create additional tablespaces for your application data. The following sample script creates some additional tablespaces:

```
CREATE TABLESPACE apps_tbs LOGGING
  DATAFILE '/u01/app/oracle/oradata/mynewdb/apps01.dbf'
  SIZE 500M REUSE AUTOEXTEND ON NEXT 1280K MAXSIZE UNLIMITED
  EXTENT MANAGEMENT LOCAL;
-- create a tablespace for indexes, separate from user tablespace (optional)
CREATE TABLESPACE indx_tbs LOGGING
  DATAFILE '/u01/app/oracle/oradata/mynewdb/indx01.dbf'
  SIZE 100M REUSE AUTOEXTEND ON NEXT 1280K MAXSIZE UNLIMITED
  EXTENT MANAGEMENT LOCAL;
```

Step 11: Run Scripts to Build Data Dictionary Views
Run the scripts necessary to build data dictionary views, synonyms, and PL/SQL packages, and to support proper functioning of SQL*Plus:

@?/rdbms/admin/catalog.sql
@?/rdbms/admin/catproc.sql
@?/sqlplus/admin/pupbld.sql
EXIT

The at-sign (@) is shorthand for the command that runs a SQL*Plus script. The question mark (?) is a SQL*Plus variable indicating the Oracle home directory. The following table contains descriptions of the scripts:

<table>
<thead>
<tr>
<th>Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG.SQL</td>
<td>Creates the views of the data dictionary tables, the dynamic performance views, and public synonyms for many of the views. Grants PUBLIC access to the synonyms.</td>
</tr>
<tr>
<td>CATPROC.SQL</td>
<td>Runs all scripts required for or used with PL/SQL.</td>
</tr>
<tr>
<td>PUPBLD.SQL</td>
<td>Required for SQL<em>Plus. Enables SQL</em>Plus to disable commands by user.</td>
</tr>
</tbody>
</table>

Step 12: Run Scripts to Install Additional Options (Optional)
You may want to run other scripts. The scripts that you run are determined by the features and options you choose to use or install. Many of the scripts available to you are described in the Oracle Database Reference. If you plan to install other Oracle products to work with this database, see the installation instructions for those products. Some products require you to create additional data dictionary tables. Usually, command files are provided to create and load these tables into the data dictionary database.

Step 13: Back Up the Database.
Take a full backup of the database to ensure that you have a complete set of files from which to recover if a media failure occurs. For information on backing up a database.

Step 14: (Optional) Enable Automatic Instance Startup
You might want to configure the Oracle instance to start automatically when its host computer restarts. For example, on Windows, use the following command to configure the database service to start the instance upon computer restart:

```
ORADIM -EDIT -SID sid -STARTMODE AUTO -SRVCSTART SYSTEM [-SPFILE]
```

You must use the -SPFILE argument if you want the instance to read an SPFILE upon automatic restart.

### 2.3. Understanding the CREATE DATABASE Statement

When you execute a CREATE DATABASE statement, Oracle Database performs a number of operations. The actual operations performed depend on the clauses that you specify in the CREATE DATABASE statement and the initialization parameters that you have set. Oracle Database performs at least these operations:

- Creates the datafiles for the database
- Creates the control files for the database
- Creates the redo log files for the database and establishes the ARCHIVELOG mode.
- Creates the SYSTEM tablespace
- Creates the SYSAUX tablespace
- Creates the data dictionary
- Sets the character set that stores data in the database
- Sets the database time zone
- Mounts and opens the database for use
- Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM

The clauses of the CREATE DATABASE statement used for specifying the passwords for users SYS and SYSTEM are:
USER SYS IDENTIFIED BY password
USER SYSTEM IDENTIFIED BY password

If you omit these clauses, these users are assigned the default passwords change_on_install and manager, respectively. A record is written to the alert log indicating that the default passwords were used. To protect your database, you must change these passwords using the ALTER USER statement immediately after database creation. Oracle strongly recommends that you specify these clauses, even though they are optional in this release of Oracle Database. The default passwords are commonly known, and if you neglect to change them later, you leave database vulnerable to attack by malicious users. When choosing a password, keep in mind that beginning in Release 11g, passwords are case sensitive. Also, there may be password formatting requirements for your database.

2.3.1. Creating a Locally Managed SYSTEM Tablespace

Specify the EXTENT MANAGEMENT LOCAL clause in the CREATE DATABASE statement to create a locally managed SYSTEM tablespace. The COMPATIBLE initialization parameter must be set to 10.0.0 or higher for this statement to be successful. If you do not specify the EXTENT MANAGEMENT LOCAL clause, by default the database creates a dictionary-managed SYSTEM tablespace. Dictionary-managed tablespaces are deprecated. If you create your database with a locally managed SYSTEM tablespace, and if you are not using Oracle-managed files, ensure that the following conditions are met: You specify the DEFAULT TEMPORARY TABLESPACE clause in the CREATE DATABASE statement. You include the UNDO TABLESPACE clause in the CREATE DATABASE statement.

2.3.2. About the SYSAUX Tablespace

The SYSAUX tablespace is always created at database creation. The SYSAUX tablespace serves as an auxiliary tablespace to the SYSTEM tablespace. Because it is the default tablespace for many Oracle Database features and products that previously required their own tablespaces, it reduces the number of tablespaces required by the database. It also reduces the load on the SYSTEM tablespace. You can specify only datafile attributes for the SYSAUX tablespace, using the SYSAUX DATAFILE clause in the CREATE DATABASE statement. Mandatory attributes of the SYSAUX tablespace are set by Oracle Database and include:

- PERMANENT
- READ WRITE
- EXTENT MANAGEMENT LOCAL
- SEGMENT SPACE MANAGEMENT AUTO

You cannot alter these attributes with an ALTER TABLESPACE statement, and any attempt to do so will result in an error. You cannot drop or rename the SYSAUX tablespace. The size of the SYSAUX tablespace is determined by the size of the database components that occupy SYSAUX. Based on the initial sizes of these components, the SYSAUX tablespace needs to be at least 240 MB at the time of database creation. The space requirements of the SYSAUX tablespace will increase after the database is fully deployed, depending on the nature of its use and workload. If you include a DATAFILE clause for the SYSTEM tablespace, then you must specify the SYSAUX DATAFILE clause as well, or the CREATE DATABASE statement will fail. This requirement does not exist if the Oracle-managed files feature is enabled. The SYSAUX tablespace has the same security attributes as the SYSTEM tablespace.

2.3.3. Using Automatic Undo Management: Creating an Undo Tablespace

Automatic undo management uses an undo tablespace. To enable automatic undo management, set the UNDO_MANAGEMENT initialization parameter to AUTO in your initialization parameter file. Or, omit this parameter, and the database defaults to automatic undo management. In this mode, undo data is stored in an undo tablespace and is managed by Oracle Database. If you want to define and name the undo tablespace yourself, you must include the UNDO TABLESPACE clause in the CREATE DATABASE statement at database creation time. If you omit this clause, and automatic undo management is enabled, the database creates a default undo tablespace named SYS_UNDOTBS.
2.3.4. Creating a Default Permanent Tablespace

The DEFAULT TABLESPACE clause of the CREATE DATABASE statement specifies a default permanent

2.3.5. Creating a Default Temporary Tablespace

The DEFAULT TEMPORARY TABLESPACE clause of the CREATE DATABASE statement creates a default
temporary tablespace for the database. Oracle Database assigns this tablespace as the temporary tablespace for
users who are not explicitly assigned a temporary tablespace. You can explicitly assign a temporary tablespace or
tablespace group to a user in the CREATE USER statement. However, if you do not do so, and if no default
temporary tablespace has been specified for the database, then by default these users are assigned the SYSTEM
tablespace as their temporary tablespace. It is not good practice to store temporary data in the SYSTEM
tablespace, and it is cumbersome to assign every user a temporary tablespace individually. Therefore, Oracle

Note: When you specify a locally managed SYSTEM tablespace, the SYSTEM tablespace cannot be used as a
temporary tablespace. In this case you must create a default temporary tablespace.

2.4. Specifying Oracle-Managed Files at Database Creation

You can minimize the number of clauses and parameters that you specify in your CREATE DATABASE statement
by using the Oracle-managed files feature. You do this by specifying either a directory or Automatic Storage
Management (ASM) disk group in which your files are created and managed by Oracle Database. By including any
of the initialization parameters DB_CREATE_FILE_DEST, DB_CREATE_ONLINE_LOG_DEST_n, or
DB_RECOVERY_FILE_DEST in your initialization parameter file, you instruct Oracle Database to create and
manage the underlying operating system files of your database. Oracle Database will automatically create and
manage the operating system files for the following database structures, depending on which initialization
parameters you specify and how you specify clauses in your CREATE DATABASE statement:

- Tablespaces and their datafiles
- Temporary tablespaces and their tempfiles
- Control files
- Redo log files
- Archived redo log files
- Flashback logs
- Block change tracking files
- RMAN backups

The following CREATE DATABASE statement shows briefly how the Oracle-managed files feature works, assuming
you have specified required initialization parameters:

CREATE DATABASE mynewdb
    USER SYS IDENTIFIED BY sys_password
    USER SYSTEM IDENTIFIED BY system_password
    EXTENT MANAGEMENT LOCAL
    UNDO TABLESPACE undotbs
    DEFAULT TEMPORARY TABLESPACE tempts1
    DEFAULT TABLESPACE users;
• The SYSTEM tablespace is created as a locally managed tablespace. Without the EXTENT MANAGEMENT LOCAL clause, the SYSTEM tablespace is created as dictionary managed, which is not recommended.

• No DATAFILE clause is specified, so the database creates an Oracle-managed datafile for the SYSTEM tablespace.

• No LOGFILE clauses are included, so the database creates two Oracle-managed redo log file groups.

• No SYSAUX DATAFILE is included, so the database creates an Oracle-managed datafile for the SYSAUX tablespace.

• No DATAFILE subclause is specified for the UNDO TABLESPACE and DEFAULT TABLESPACE clauses, so the database creates an Oracle-managed datafile for each of these tablespaces.

• No TEMPFILE subclause is specified for the DEFAULT TEMPORARY TABLESPACE clause, so the database creates an Oracle-managed tempfile.

• If no CONTROL_FILES initialization parameter is specified in the initialization parameter file, then the database also creates an Oracle-managed control file.

### 2.4.1. Supporting Bigfile Tablespaces during Database Creation

Oracle Database simplifies management of tablespaces and enables support for ultra-large databases by letting you create bigfile tablespaces. Bigfile tablespaces can contain only one file, but that file can have up to 4G blocks. The maximum number of datafiles in an Oracle Database is limited (usually to 64K files). Therefore, bigfile tablespaces can significantly enhance the storage capacity of an Oracle Database. This section discusses the clauses of the CREATE DATABASE statement that let you include support for bigfile tablespaces.

### 2.4.2. Overriding the Default Tablespace Type

The SYSTEM and SYSAUX tablespaces are always created with the default tablespace type. However, you can explicitly override the default tablespace type for the UNDO and DEFAULT TEMPORARY tablespace during the CREATE DATABASE operation. For example, you can create a bigfile UNDO tablespace in a database with the default tablespace type of smallfile as follows:

```sql
CREATE DATABASE mynewdb
...
  BIGFILE UNDO TABLESPACE undotbs
    DATAFILE '/u01/oracle/oradata/mynewdb/undotbs01.dbf'
    SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

You can create a smallfile DEFAULT TEMPORARY tablespace in a database with the default tablespace type of bigfile as follows:

```sql
CREATE DATABASE mynewdb
  SET DEFAULT BIGFILE TABLSPACE
...
  SMALLFILE DEFAULT TEMPORARY TABLESPACE tempts1
    TEMPFILE '/u01/oracle/oradata/mynewdb/temp01.dbf'
    SIZE 20M REUSE
```

### Specifying FORCE LOGGING Mode

Some data definition language statements (such as CREATE TABLE) allow the NOLOGGING clause, which causes some database operations not to generate redo records in the database redo log. The NOLOGGING setting can speed up operations that can be easily recovered outside of the database recovery mechanisms, but it can negatively affect media recovery and standby databases.

Oracle Database lets you force the writing of redo records even when NOLOGGING has been specified in DDL
Oracle statements. The database never generates redo records for temporary tablespaces and temporary segments, so forced logging has no affect for objects.

Using the FORCE LOGGING Clause
To put the database into FORCE LOGGING mode, use the FORCE LOGGING clause in the CREATE DATABASE statement. If you do not specify this clause, the database is not placed into FORCE LOGGING mode. Use the ALTER DATABASE statement to place the database into FORCE LOGGING mode after database creation. This statement can take a considerable time for completion, because it waits for all unlogged direct writes to complete. You can cancel FORCE LOGGING mode using the following SQL statement:

```sql
ALTER DATABASE NO FORCE LOGGING;
```

Independent of specifying FORCE LOGGING for the database, you can selectively specify FORCE LOGGING or NO FORCE LOGGING at the tablespace level. However, if FORCE LOGGING mode is in effect for the database, it takes precedence over the tablespace setting. If it is not in effect for the database, then the individual tablespace settings are enforced. Oracle recommends that either the entire database is placed into FORCE LOGGING mode, or individual tablespaces be placed into FORCE LOGGING mode, but not both. The FORCE LOGGING mode is a persistent attribute of the database. That is, if the database is shut down and restarted, it remains in the same logging mode. However, if you re-create the control file, the database is not restarted in the FORCE LOGGING mode unless you specify the FORCE LOGGING clause in the CREATE CONTROL FILE statement.

Performance Considerations of FORCE LOGGING Mode
FORCE LOGGING mode results in some performance degradation. If the primary reason for specifying FORCE LOGGING is to ensure complete media recovery, and there is no standby database active, then consider the following:

- How many media failures are likely to happen?
- How serious is the damage if unlogged direct writes cannot be recovered?
- Is the performance degradation caused by forced logging tolerable?

If the database is running in NOARCHIVELOG mode, then generally there is no benefit to placing the database in FORCE LOGGING mode. Media recovery is not possible in NOARCHIVELOG mode, so if you combine it with FORCE LOGGING, the result may be performance degradation with little benefit.

2.5. Understanding Initialization Parameters

When an Oracle instance starts, it reads initialization parameters from an initialization parameter file. For any initialization parameters not specifically included in the initialization parameter file, the database supplies defaults. The initialization parameter file can be either a read-only text file, or a read/write binary file. The binary file is called a server parameter file. A server parameter file enables you to change initialization parameters with ALTER SYSTEM commands and to persist the changes across a shutdown and startup. It also provides a basis for self-tuning by Oracle Database. For these reasons, it is recommended that you use a server parameter file. You can create one manually from your edited text initialization file, or automatically by using Database Configuration Assistant (DBCA) to create your database.

Before you manually create a server parameter file, you can start an instance with a text initialization parameter file. Upon startup, the Oracle instance first searches for a server parameter file in a default location, and if it does not find one, searches for a text initialization parameter file. You can also override an existing server parameter file by naming a text initialization parameter file as an argument of the STARTUP command.

2.5.1. Sample Initialization Parameter File

Oracle Database provides generally appropriate values in a sample text initialization parameter file. You can edit these Oracle-supplied initialization parameters and add others, depending upon your configuration and options and how you plan to tune the database.
The sample text initialization parameter file is named init.ora and is found in the following location on most platforms:

```
ORACLE_HOME/dbs
```

The following is the content of the sample file:

```
##############################################################
# Example INIT.ORA file
#
# This file is provided by Oracle Corporation to help you start by providing
# a starting point to customize your RDBMS installation for your site.
#
# NOTE: The values that are used in this file are only intended to be used
# as a starting point. You may want to adjust/tune those values to your
# specific hardware and needs. You may also consider using Database
# Configuration Assistant tool (DBCA) to create INIT file and to size your
# initial set of tablespaces based on the user input.
##############################################################

# Change '<ORACLE_BASE>' to point to the oracle base (the one you specify at
# install time)

db_name='ORCL'
memory_target=1G
processes = 150
audit_file_dest='<ORACLE_BASE>/admin/orcl/adump'
audit_trail = 'db'
db_block_size=8192
db_domain=''
db_recovery_file_dest='<ORACLE_BASE>/flash_recovery_area'
db_recovery_file_dest_size=2G
diagnostic_dest='<ORACLE_BASE>'
dispatchers='(PROTOCOL=TCP) (SERVICE=ORCLXDB)'
open_cursors=300
remote_login_passwordfile='EXCLUSIVE'
do undotablespace='UNDOTBS1'

# You may want to ensure that control files are created on separate physical
# devices
control_files = (ora_control1, ora_control2)
compatible = '11.1.0'
```

If you are creating an Oracle Database for the first time, Oracle suggests that you minimize the number of parameter values that you alter. As you become more familiar with your database and environment, you can dynamically tune many initialization parameters using the ALTER SYSTEM statement. If you are using a text initialization parameter file, your changes are effective only for the current instance. To make them permanent, you must update them manually in the initialization parameter file, or they will be lost over the next shutdown and startup of the database. If you are using a server parameter file, initialization parameter file changes made by the ALTER SYSTEM statement can persist across shutdown and startup. This section introduces you to some of the basic initialization parameters you can add or edit before you create your new database.

**Determining the Global Database Name**

The global database name consists of the user-specified local database name and the location of the database within a network structure. The DB_NAME initialization parameter determines the local name component of the database name, and the DB_DOMAIN parameter, which is optional, indicates the domain (logical location) within a network structure. The combination of the settings for these two parameters must form a database name that is unique within a network. For example, to create a database with a global database name of test.us.acme.com, edit the parameters of the new parameter file as follows:
DB_NAME = test
DB_DOMAIN = us.acme.com

You can rename the GLOBAL_NAME of your database using the ALTER DATABASE RENAME GLOBAL_NAME statement. However, you must also shut down and restart the database after first changing the DB_NAME and DB_DOMAIN initialization parameters and recreating the control files. Recreating the control files is easily accomplished with the command ALTER DATABASE BACKUP CONTROLFILE TO TRACE.

**DB_NAME Initialization Parameter**

DB_NAME must be set to a text string of no more than eight characters. During database creation, the name provided for DB_NAME is recorded in the datafiles, redo log files, and control file of the database. If during database instance startup the value of the DB_NAME parameter (in the parameter file) and the database name in the control file are not the same, the database does not start.

**DB_DOMAIN Initialization Parameter**

DB_DOMAIN is a text string that specifies the network domain where the database is created. If the database you are about to create will ever be part of a distributed database system, give special attention to this initialization parameter before database creation. This parameter is optional.

**Specifying a Flash Recovery Area**

A flash recovery area is a location in which Oracle Database can store and manage files related to backup and recovery. It is distinct from the database area, which is a location for the current database files (datafiles, control files, and online redo logs). You specify a flash recovery area with the following initialization parameters:

- **DB_RECOVERY_FILE_DEST**: Location of the flash recovery area. This can be a directory, file system, or Automatic Storage Management (ASM) disk group. It cannot be a raw file system. In an Oracle Real Application Clusters (RAC) environment, this location must be on a cluster file system, ASM disk group, or a shared directory configured through NFS.

- **DB_RECOVERY_FILE_DEST_SIZE**: Specifies the maximum total bytes to be used by the flash recovery area. This initialization parameter must be specified before DB_RECOVERY_FILE_DEST is enabled. In an Oracle RAC environment, the settings for these two parameters must be the same on all instances. You cannot enable these parameters if you have set values for the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameters. You must disable those parameters before setting up the flash recovery area. You can instead set values for the LOG_ARCHIVE_DEST_n parameters. If you do not set values for local LOG_ARCHIVE_DEST_n, then setting up the flash recovery area will implicitly set LOG_ARCHIVE_DEST_10 to the flash recovery area. Oracle recommends using a flash recovery area, because it can simplify backup and recovery operations for your database.

**Specifying Control Files**

The CONTROL_FILES initialization parameter specifies one or more control filenames for the database. When you execute the CREATE DATABASE statement, the control files listed in the CONTROL_FILES parameter are created. If you do not include CONTROL_FILES in the initialization parameter file, then Oracle Database creates a control file in the same directory as the initialization parameter file, using a default operating system–dependent filename. If you have enabled Oracle-managed files, the database creates Oracle-managed control files. If you want the database to create new operating system files when creating database control files, the filenames listed in the CONTROL_FILES parameter must not match any filenames that currently exist on your system. If you want the database to reuse or overwrite existing files when creating database control files, ensure that the filenames listed in the CONTROL_FILES parameter match the filenames that are to be reused, and include a CONTROLFILE REUSE clause in the CREATE DATABASE statement. Oracle strongly recommends you use at least two control files stored on separate physical disk drives for each database.

**Specifying Database Block Sizes**

The DB_BLOCK_SIZE initialization parameter specifies the standard block size for the database. This block size is used for the SYSTEM tablespace and by default in other tablespaces. Oracle Database can support up to four additional nonstandard block sizes.
**DB_BLOCK_SIZE Initialization Parameter**

The most commonly used block size should be picked as the standard block size. In many cases, this is the only block size that you need to specify. Typically, `DB_BLOCK_SIZE` is set to either 4K or 8K. If you do not set a value for this parameter, the default data block size is operating system specific, which is generally adequate. You cannot change the block size after database creation except by re-creating the database. If the database block size is different from the operating system block size, ensure that the database block size is a multiple of the operating system block size. For example, if your operating system block size is 2K (2048 bytes), the following setting for the `DB_BLOCK_SIZE` initialization parameter is valid:

```
DB_BLOCK_SIZE=4096
```

A larger data block size provides greater efficiency in disk and memory I/O (access and storage of data). Therefore, consider specifying a block size larger than your operating system block size if the following conditions exist:

- Oracle Database is on a large computer system with a large amount of memory and fast disk drives. For example, databases controlled by mainframe computers with vast hardware resources typically use a data block size of 4K or greater.
- The operating system that runs Oracle Database uses a small operating system block size. For example, if the operating system block size is 1K and the default data block size matches this, the database may be performing an excessive amount of disk I/O during normal operation. For best performance in this case, a database block should consist of multiple operating system blocks.

**Nonstandard Block Sizes**

Tablespaces of nonstandard block sizes can be created using the `CREATE TABLESPACE` statement and specifying the `BLOCKSIZE` clause. These nonstandard block sizes can have any of the following power-of-two values: 2K, 4K, 8K, 16K or 32K. Platform-specific restrictions regarding the maximum block size apply, so some of these sizes may not be allowed on some platforms. To use nonstandard block sizes, you must configure subcaches within the buffer cache area of the SGA memory for all of the nonstandard block sizes that you intend to use. The ability to specify multiple block sizes for your database is especially useful if you are transporting tablespaces between databases. You can, for example, transport a tablespace that uses a 4K block size from an OLTP environment to a data warehouse environment that uses a standard block size of 8K.

**Specifying the Maximum Number of Processes**

The `PROCESSES` initialization parameter determines the maximum number of operating system processes that can be connected to Oracle Database concurrently. The value of this parameter must be a minimum of one for each background process plus one for each user process. The number of background processes will vary according to the database features that you are using. For example, if you are using Advanced Queuing or the file mapping feature, you will have additional background processes. If you are using Automatic Storage Management, then add three additional processes for the database instance. If you plan on running 50 user processes, a good estimate would be to set the `PROCESSES` initialization parameter to 70.

**Specifying the DDL Lock Timeout**

Data Definition Language (DDL) statements require exclusive locks on internal structures. If these locks are unavailable when a DDL statement runs, the DDL statement fails, though it might have succeeded if it had been executed sub seconds later. To enable DDL statements to wait for locks, specify a DDL lock timeout—the number of seconds a DDL command waits for its required locks before failing. To specify a DDL lock timeout, use the `DDL_LOCK_TIMEOUT` parameter. The permissible range of values for `DDL_LOCK_TIMEOUT` is 0 to 100,000. The default is 0. You can set `DDL_LOCK_TIMEOUT` at the system level, or at the session level with an `ALTER SESSION` statement.

**Specifying the Method of Undo Space Management**

Every Oracle Database must have a method of maintaining information that is used to undo changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. Collectively these records are called undo data.

**UNDO_MANAGEMENT Initialization Parameter**

The `UNDO_MANAGEMENT` initialization parameter determines whether or not an instance starts in automatic undo management mode, which stores undo in an undo tablespace. Set this parameter to `AUTO` to enable automatic undo management mode. Beginning with Release 11g, `AUTO` is the default if the parameter is omitted or is null.
UNDO_TABLESPACE Initialization Parameter
When an instance starts up in automatic undo management mode, it attempts to select an undo tablespace for storage of undo data. If the database was created in automatic undo management mode, then the default undo tablespace (either the system-created SYS_UNDOTBS tablespace or the user-specified undo tablespace) is the undo tablespace used at instance startup. You can override this default for the instance by specifying a value for the UNDO_TABLESPACE initialization parameter. This parameter is especially useful for assigning a particular undo tablespace to an instance in an Oracle Real Application Clusters environment. If no undo tablespace is specified by the UNDO_TABLESPACE initialization parameter, then the first available undo tablespace in the database is chosen. If no undo tablespace is available, then the instance starts without an undo tablespace, and undo data is written to the SYSTEM tablespace. You should avoid running in this mode.

Note: When using the CREATE DATABASE statement to create a database, do not include a UNDO_TABLESPACE parameter in the initialization parameter file. Instead, include an UNDO_TABLESPACE clause in the CREATE DATABASE statement.

About The COMPATIBLE Initialization Parameter
The COMPATIBLE initialization parameter enables or disables the use of features in the database that affect file format on disk. For example, if you create an Oracle Database 11g Release 1 (11.1) database, but specify COMPATIBLE = 10.0.0 in the initialization parameter file, then features that requires 11.1 compatibility generate an error if you try to use them. Such a database is said to be at the 10.0.0 compatibility level. You can advance the compatibility level of your database. If you do advance the compatibility of your database with the COMPATIBLE initialization parameter, there is no way to start the database using a lower compatibility level setting, except by doing a point-in-time recovery to a time before the compatibility was advanced. The default value for the COMPATIBLE parameter is the release number of the most recent major release.

Note: For Oracle Database 11g Release 1 (11.1), the default value of the COMPATIBLE parameter is 11.1.0. The minimum value is 10.0.0. If you create an Oracle Database using the default value, you can immediately use all the new features in this release, and you can never downgrade the database.

2.6. Dropping a Database
Dropping a database involves removing its datafiles; redo log files, control files, and initialization parameter files. The DROP DATABASE statement deletes all control files and all other database files listed in the control file. To use the DROP DATABASE statement successfully, all of the following conditions must apply:

- The database must be mounted and closed.
- The database must be mounted exclusively—not in shared mode.
- The database must be mounted as RESTRICTED.

An example of this statement is:

DROP DATABASE;

The DROP DATABASE statement has no effect on archived log files, nor does it have any effect on copies or backups of the database. It is best to use RMAN to delete such files. If the database is on raw disks, the actual raw disk special files are not deleted. If you used the Database Configuration Assistant to create your database, you can use that tool to delete (drop) your database and remove the files.

2.7. Managing Initialization Parameters Using a Server Parameter File
Initialization parameters for the Oracle Database have traditionally been stored in a text initialization parameter file. For better manageability, you can choose to maintain initialization parameters in a binary server parameter file that is persistent across database startup and shutdown. This section introduces the server parameter file, and explains how to manage initialization parameters using either method of storing the parameters.
2.7.1. What Is a Server Parameter File?

A server parameter file can be thought of as a repository for initialization parameters that is maintained on the machine running the Oracle Database server. It is, by design, a server-side initialization parameter file. Initialization parameters stored in a server parameter file are persistent, in that any changes made to the parameters while an instance is running can persist across instance shutdown and startup. This arrangement eliminates the need to manually update initialization parameters to make persistent any changes effected by ALTER SYSTEM statements. It also provides a basis for self-tuning by the Oracle Database server. A server parameter file is initially built from a text initialization parameter file using the CREATE SPFILE statement. (It can also be created directly by the Database Configuration Assistant.) The server parameter file is a binary file that cannot be edited using a text editor. Oracle Database provides other interfaces for viewing and modifying parameter settings in a server parameter file.

Caution: Although you can open the binary server parameter file with a text editor and view its text, do not manually edit it. Doing so will corrupt the file. You will not be able to start your instance, and if the instance is running, it could fail. When you issue a STARTUP command with no PFILE clause, the Oracle instance searches an operating system–specific default location for a server parameter file from which to read initialization parameter settings. If no server parameter file is found, the instance searches for a text initialization parameter file. If a server parameter file exists but you want to override it with settings in a text initialization parameter file, you must specify the PFILE clause when issuing the STARTUP command.

2.7.2. Migrating to a Server Parameter File

If you are currently using a text initialization parameter file, use the following steps to migrate to a server parameter file.

1. If the initialization parameter file is located on a client machine, transfer the file (for example, FTP) from the client machine to the server machine.

   Note: If you are migrating to a server parameter file in an Oracle Real Application Clusters environment, you must combine all of your instance-specific initialization parameter files into a single initialization parameter file.

2. Create a server parameter file in the default location using the CREATE SPFILE FROM PFILE statement. This statement reads the text initialization parameter file to create a server parameter file. The database does not have to be started to issue a CREATE SPFILE statement.

3. Start up or restart the instance.
   The instance finds the new SPFILE in the default location and starts up with it.

2.7.3. Creating a Server Parameter File

You use the CREATE SPFILE statement to create a server parameter file. You must have the SYSDBA or the SYSOPER system privilege to execute this statement.

Note: When you use the Database Configuration Assistant to create a database, it automatically creates a server parameter file for you.

The CREATE SPFILE statement can be executed before or after instance startup. However, if the instance has been started using a server parameter file, an error is raised if you attempt to re-create the same server parameter file that is currently being used by the instance. You can create a server parameter file (SPFILE) from an existing text initialization parameter file or from memory. Creating the SPFILE from memory means copying the current values of initialization parameters in the running instance to the SPFILE. The following example creates a server parameter file from text initialization parameter file /u01/oracle/dbs/init.ora. In this example no SPFILE name is specified.

```
CREATE SPFILE FROM PFILE='/u01/oracle/dbs/init.ora';
```

The next example illustrates creating a server parameter file and supplying a name and location.

```
CREATE SPFILE='/'u01/oracle/dbs/test_spfile.ora' FROM
```
PFILE='/u01/oracle/dbs/test_init.ora';

The next example illustrates creating a server parameter file in the default location from the current values of the initialization parameters in memory.

```
CREATE SPFILE FROM MEMORY;
```

Whether you use the default SPFILE name and default location or specify an SPFILE name and location, if an SPFILE of the same name already exists in the location, it is overwritten without a warning message. When you create an SPFILE from a text initialization parameter file, comments specified on the same lines as a parameter setting in the initialization parameter file are maintained in the SPFILE. All other comments are ignored. Oracle recommends that you allow the database to give the SPFILE the default name and store it in the default location. This eases administration of your database. For example, the STARTUP command assumes this default location to read the SPFILE.

Note: Upon startup, the instance first searches for an SPFILE named spfileORACLE_SID.ora, and if not found, searches for spfile.ora. Using spfile.ora enables all Real Application Cluster (RAC) instances to use the same server parameter file. If neither SPFILE is found, the instance searches for the text initialization parameter file initORACLE_SID.ora. If you create an SPFILE in a location other than the default location, you must create a text initialization parameter file that points to the server parameter file.

When Installing or Initially Creating a Release 11g Database

When first installing or creating a Release 11g database, the COMPATIBLE initialization parameter defaults to 11.1.0, so this requirement for a HARD-compliant server parameter file (SPFILE) is met. You must then ensure that the SPFILE is stored on HARD-enabled storage. To meet this requirement, do one of the following:

For an Oracle Real Application Clusters environment without shared storage, when DBCA prompts for the location of the SPFILE, specify a location on HARD-enabled storage.

For a single-instance installation, or for an Oracle Real Application Clusters environment with shared storage, complete these steps:

1. Complete the database installation with Database Configuration Assistant (DBCA).
   The SPFILE is created in the default location.
2. Do one of the following:
   Using an operating system command, copy the SPFILE to HARD-enabled storage.
   In SQL*Plus or another interactive environment such as SQL Developer, connect to the database as user SYS and then submit the following command:

   ```
   CREATE SPFILE = 'spfile_name' FROM MEMORY;
   ```

   Where spfile_name is a complete path name, including file name that points to HARD-enabled storage.
3. Do one of the following:
   Create a text initialization parameter file (PFILE) in the default location with the following single entry:

   ```
   SPFILE = spfile_name
   ```

   Where spfile_name is the complete path to the SPFILE on HARD-enabled storage.
   On the UNIX and Linux platforms, in the default SPFILE location, create a symbolic link to the SPFILE on HARD-enabled storage.
4. Shut down the database instance.
5. Delete the SPFILE in the default location.
6. Start up the database instance.

When Upgrading to Release 11g from an Earlier Database Release

When upgrading to Release 11g from an earlier database release, complete these steps to migrate your SPFILE to
the HARD-compliant format and to store the SPFILE on HARD-enabled storage:  
Start SQL*Plus or another interactive query application, log in to the database as user SYS or SYSTEM, and then enter the following command:

```
ALTER SYSTEM SET COMPATIBLE = '11.1.0' SCOPE = SPFILE;
```

**Warning:** Advancing the compatibility level to 11.1.0 enables Release 11g features and file formats and has wide repercussions. Consult Oracle Database Upgrade Guide before proceeding.

2. Restart the database instance.  
The database is now at compatibility level 11.1.0.  
3. If your SPFILE is not already on HARD-enabled storage, complete the following steps:  
a. In SQL*Plus or another interactive environment, connect to the database as user SYS and then submit the following command:

```
CREATE SPFILE = 'spfile_name' FROM MEMORY;
```

Where `spfile_name` is a complete path name, including file name that points to HARD-enabled storage.  
b. Do one of the following:  
Create a text initialization parameter file (PFILE) in the default location with the following single entry:

```
SPFILE = spfile_name
```

Where `spfile_name` is the complete path to the SPFILE on HARD-enabled storage.  
On the UNIX and Linux platforms, in the default SPFILE location, create a symbolic link to the SPFILE on HARD-enabled storage.  
c. Shut down the database instance.  
d. Delete the SPFILE in the default location.  
e. Start up the database instance.  

### 2.7.3. The SPFILE Initialization Parameter

The SPFILE initialization parameter contains the name of the current server parameter file. When the default server parameter file is used by the database—that is, you issue a STARTUP command and do not specify a PFILE parameter—the value of SPFILE is internally set by the server. The SQL*Plus command SHOW PARAMETERS SPFILE (or any other method of querying the value of a parameter) displays the name of the server parameter file that is currently in use.

**Changing Initialization Parameter Values**

The ALTER SYSTEM statement enables you to set, change, or restore to default the values of initialization parameters. If you are using a text initialization parameter file, the ALTER SYSTEM statement changes the value of a parameter only for the current instance, because there is no mechanism for automatically updating text initialization parameters on disk. You must update them manually to be passed to a future instance. Using a server parameter file overcomes this limitation. There are two kinds of initialization parameters:

- Dynamic initialization parameters can be changed for the current Oracle Database instance. The changes take effect immediately.
- Static initialization parameters cannot be changed for the current instance. You must change these parameters in the text initialization file or server parameter file and then restart the database before changes take effect.

**Setting or Changing Initialization Parameter Values**

Use the SET clause of the ALTER SYSTEM statement to set or change initialization parameter values. The optional SCOPE clause specifies the scope of a change as described in the following table:
SCOPE Clause Description

SCOPE = SPFILE The change is applied in the server parameter file only. The effect is as follows:
No change is made to the current instance.
For both dynamic and static parameters, the change is effective at the next startup and is persistent.
This is the only SCOPE specification allowed for static parameters.

SCOPE = MEMORY The change is applied in memory only. The effect is as follows:
- The change is made to the current instance and is effective immediately.
- For dynamic parameters, the effect is immediate, but it is not persistent because the server parameter file
  is not updated. For static parameters, this specification is not allowed.

SCOPE = BOTH The change is applied in both the server parameter file and memory. The effect is as
follows:
- The change is made to the current instance and is effective immediately.
- For dynamic parameters, the effect is persistent because the server parameter file is updated.
- For static parameters, this specification is not allowed.

It is an error to specify SCOPE=SPFILE or SCOPE=BOTH if the instance did not start up with a server parameter
file. The default is SCOPE=BOTH if a server parameter file was used to start up the instance, and MEMORY if a
text initialization parameter file was used to start up the instance. For dynamic parameters, you can also specify the
DEFERRED keyword. When specified, the change is effective only for future sessions. When you specify SCOPE
as SPFILE or BOTH, an optional COMMENT clause lets you associate a text string with the parameter update. The
comment is written to the server parameter file.

The following statement changes the maximum number of failed login attempts before the connection is dropped. It
includes a comment, and explicitly states that the change is to be made only in the server parameter file.

```
ALTER SYSTEM SET SEC_MAX_FAILED_LOGIN_ATTEMPTS=3 COMMENT='Reduce from 10 for tighter security.' SCOPE=SPFILE;
```

The next example sets a complex initialization parameter that takes a list of attributes. Specifically, the parameter
value being set is the LOG_ARCHIVE_DEST_n initialization parameter. This statement could change an existing
setting for this parameter or create a new archive destination.

```
ALTER SYSTEM SET
LOG_ARCHIVE_DEST_4='LOCATION=/u02/oracle/rbdb1/',MANDATORY,'REOPEN=2'
COMMENT='Add new destination on Nov 29' SCOPE=SPFILE;
```

When a value consists of a list of parameters, you cannot edit individual attributes by the position or ordinal number.
You must specify the complete list of values each time the parameter is updated, and the new list completely
replaces the old list.

Clearing Initialization Parameter Values

You can use the ALTER SYSTEM RESET command to clear (remove) the setting of any initialization parameter in
the SPFILE that was used to start the instance. Neither SCOPE=MEMORY nor SCOPE=BOTH are allowed. The
SCOPE = SPFILE clause is not required, but can be included.
You may want to clear a parameter in the SPFILE so that upon the next database startup a default value is used.

2.7.4. Recovering a lost or damaged spfile

If your server parameter file (SPFILE) becomes lost or corrupted, the current instance may fail, or the next attempt at
starting the database instance may fail. There are a number of ways to recover the SPFILE:
If the instance is running, issue the following command to recreate the SPFILE from the current values of
initialization parameters in memory:

```
CREATE SPFILE FROM MEMORY;
```
This command creates the SPFILE with the default name and in the default location. You can also create the SPFILE with a new name or in a specified location.

If you have a valid text initialization parameter file (PFILE), recreate the SPFILE from the PFILE with the following command:

```
CREATE SPFILE FROM PFILE;
```

This command assumes that the PFILE is in the default location and has the default name. Restore the SPFILE from backup. If none of the previous methods are possible in your situation, perform these steps:

1. Create a text initialization parameter file (PFILE) from the parameter value listings in the alert log.

   When an instance starts up, the initialization parameters used for startup are written to the alert log. You can copy and paste this section from the text version of the alert log (without XML tags) into a new PFILE.

2. Create the SPFILE from the PFILE.
3. Oracle Instance Memory Architecture

3.1. Basic Memory Structure

The basic memory structures associated with Oracle Database include:

3.1.1. System Global Area (SGA)

The SGA is a group of shared memory structures, known as SGA components that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.

3.1.2. Program Global Area (PGA)

A PGA is a memory region that contains data and control information for a server process. It is nonshared memory created by Oracle Database when a server process is started. Access to the PGA is exclusive to the server process. There is one PGA for each server process. Background processes also allocate their own PGAs. The total PGA memory allocated for all background and server processes attached to an Oracle Database instance is referred to as the total instance PGA memory and the collection of all individual PGAs is referred to as the total instance PGA, or just instance PGA. Figure illustrates the relationships among these memory structures.

3.1.3. Setting the Buffer Cache Initialization Parameters

The buffer cache initialization parameters determine the size of the buffer cache component of the SGA. You use them to specify the sizes of caches for the various block sizes used by the database. These initialization parameters are all dynamic. The size of a buffer cache affects performance. Larger cache sizes generally reduce the number of disk reads and writes. However, a large cache may take up too much memory and induce memory paging or swapping.

Oracle Database supports multiple block sizes in a database. If you create tablespaces with non-standard block sizes, you must configure non-standard block size buffers to accommodate these tablespaces. The standard block size is used for the SYSTEM tablespace. You specify the standard block size by setting the initialization parameter DB_BLOCK_SIZE. Legitimate values are from 2K to 32K. If you intend to use multiple block sizes in your database,
you must have the DB_CACHE_SIZE and at least one DB_nK_CACHE_SIZE parameter set. Oracle Database assigns an appropriate default value to the DB_CACHE_SIZE parameter, but the DB_nK_CACHE_SIZE parameters default to 0, and no additional block size caches are configured.

The sizes and numbers of non-standard block size buffers are specified by the following parameters:

- `DB_2K_CACHE_SIZE`
- `DB_4K_CACHE_SIZE`
- `DB_8K_CACHE_SIZE`
- `DB_16K_CACHE_SIZE`
- `DB_32K_CACHE_SIZE`

Each parameter specifies the size of the cache for the corresponding block size.

**Note:** Platform-specific restrictions regarding the maximum block size apply, so some of these sizes might not be allowed on some platforms.

**Example of Setting Block and Cache Sizes**

- `DB_BLOCK_SIZE=4096`
- `DB_CACHE_SIZE=1024M`
- `DB_2K_CACHE_SIZE=256M`
- `DB_8K_CACHE_SIZE=512M`

In the preceding example, the parameter `DB_BLOCK_SIZE` sets the standard block size of the database to 4K. The size of the cache of standard block size buffers is 1024MB. Additionally, 2K and 8K caches are also configured, with sizes of 256MB and 512MB, respectively.

**Notes:** The DB_nK_CACHE_SIZE parameters cannot be used to size the cache for the standard block size. If the value of `DB_BLOCK_SIZE` is nK, it is invalid to set `DB_nK_CACHE_SIZE`. The size of the cache for the standard block size is always determined from the value of `DB_CACHE_SIZE`.

The cache has a limited size, so not all the data on disk can fit in the cache. When the cache is full, subsequent cache misses cause Oracle Database to write dirty data already in the cache to disk to make room for the new data. (If a buffer is not dirty, it does not need to be written to disk before a new block can be read into the buffer.) Subsequent access to any data that was written to disk and then overwritten results in additional cache misses. The size of the cache affects the likelihood that a request for data results in a cache hit. If the cache is large, it is more likely to contain the data that is requested. Increasing the size of a cache increases the percentage of data requests that result in cache hits. You can change the size of the buffer cache while the instance is running, without having to shut down the database. Do this with the ALTER SYSTEM statement. Use the fixed view `V$BUFFER_POOL` to track the sizes of the different cache components and any pending resize operations.

**Multiple Buffer Pools:** You can configure the database buffer cache with separate buffer pools that either keep data in the buffer cache or make the buffers available for new data immediately after using the data blocks. Particular schema objects (tables, clusters, indexes, and partitions) can then be assigned to the appropriate buffer pool to control the way their data blocks age out of the cache.

- The KEEP buffer pool retains the schema object’s data blocks in memory.
- The RECYCLE buffer pool eliminates data blocks from memory as soon as they are no longer needed.
- The DEFAULT buffer pool contains data blocks from schema objects that are not assigned to any buffer pool, as well as schema objects that are explicitly assigned to the DEFAULT pool.

The initialization parameters that configure the KEEP and RECYCLE buffer pools are `DB_KEEP_CACHE_SIZE` and `DB_RECYCLE_CACHE_SIZE`. 
Note: Multiple buffer pools are only available for the standard block size. Non-standard block size caches have a single DEFAULT pool.

3.1.4. Specifying the Shared Pool Size

The SHARED_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the shared pool component of the SGA. Oracle Database selects an appropriate default value. In releases before Oracle Database 10g Release 1, the amount of shared pool memory that was allocated was equal to the value of the SHARED_POOL_SIZE initialization parameter plus the amount of internal SGA overhead computed during instance startup. The internal SGA overhead refers to memory that is allocated by Oracle Database during startup, based on the values of several other initialization parameters.

This memory is used to maintain state for different server components in the SGA. For example, if the SHARED_POOL_SIZE parameter is set to 64MB and the internal SGA overhead is computed to be 12MB, the real size of the shared pool is 64+12=76MB, although the value of the SHARED_POOL_SIZE parameter is still displayed as 64MB. Starting with Oracle Database 10g Release 1, the size of the internal SGA overhead is included in the user-specified value of SHARED_POOL_SIZE. If you are not using automatic memory management or automatic shared memory management, the amount of shared pool memory that is allocated at startup is equal to the value of the SHARED_POOL_SIZE initialization parameter, rounded up to a multiple of the granule size. You must therefore set this parameter so that it includes the internal SGA overhead in addition to the desired value for shared pool size. In the previous example, if the SHARED_POOL_SIZE parameter is set to 64MB at startup, then the available shared pool after startup is 64-12=52MB, assuming the value of internal SGA overhead remains unchanged. In order to maintain an effective value of 64MB for shared pool memory after startup, you must set the SHARED_POOL_SIZE parameter to 64+12=76MB.

When migrating from a release that is earlier than Oracle Database 10g Release 1, the Oracle Database 11g migration utilities recommend a new value for this parameter based on the value of internal SGA overhead in the pre-upgrade environment and based on the old value of this parameter. Beginning with Oracle Database 10g, the exact value of internal SGA overhead, also known as startup overhead in the shared pool, can be queried from the V$SGAINFO view. Also, in manual shared memory management mode, if the user-specified value of SHARED_POOL_SIZE is too small to accommodate even the requirements of internal SGA overhead, then Oracle Database generates an ORA-371 error during startup, with a suggested value to use for the SHARED_POOL_SIZE parameter. When you use automatic shared memory management in Oracle Database 11g, the shared pool is automatically tuned, and an ORA-371 error would not be generated.

The Result Cache and Shared Pool Size the result cache takes its memory from the shared pool. Therefore, if you expect to increase the maximum size of the result cache, take this into consideration when sizing the shared pool.

3.1.5. Specifying the Large Pool Size

The LARGE_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the large pool component of the SGA. The large pool is an optional component of the SGA. You must specifically set the LARGE_POOL_SIZE parameter if you want to create a large pool.

3.1.6. Specifying the Java Pool Size

The JAVA_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the java pool component of the SGA. Oracle Database selects an appropriate default value.

3.1.7. Specifying the Streams Pool Size

The STREAMS_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the Streams Pool component of the SGA. If STREAMS_POOL_SIZE is set to 0, then the Oracle Streams product transfers memory from the buffer cache to the Streams Pool when it is needed.

3.1.8. Specifying the Result Cache Maximum Size

The RESULT_CACHE_MAX_SIZE initialization parameter is a dynamic parameter that enables you to specify the
maximum size of the result cache component of the SGA. Typically, there is no need to specify this parameter, because the default maximum size is chosen by the database based on total memory available to the SGA and on the memory management method currently in use. You can view the current default maximum size by displaying the value of the RESULT_CACHE_MAX_SIZE parameter. If you want to change this maximum size, you can set RESULT_CACHE_MAX_SIZE with an ALTER SYSTEM statement or you can specify this parameter in the text initialization parameter file. In each case, the value is rounded up to the nearest multiple of 32K.

If RESULT_CACHE_MAX_SIZE is 0 upon instance startup, the result cache is disabled. To re-enable it you must set RESULT_CACHE_MAX_SIZE to a nonzero value (or remove this parameter from the text initialization parameter file to get the default maximum size) and then restart the database. Note that after starting the database with the result cache disabled, if you use an ALTER SYSTEM statement to set RESULT_CACHE_MAX_SIZE to a nonzero value but do not restart the database, querying the value of the RESULT_CACHE_MAX_SIZE parameter returns a nonzero value even though the result cache is still disabled. The value of RESULT_CACHE_MAX_SIZE is therefore not the most reliable way to determine if the result cache is enabled. You can use the following query instead:

```
SELECT dbms_result_cache.status() FROM dual;
```

```
DBMS_RESULT_CACHE.STATUS()
---------------------------------------------
ENABLED
```

The result cache takes its memory from the shared pool, so if you increase the maximum result cache size, consider also increasing the shared pool size. The view V$RESULT_CACHE_STATISTICS and the PL/SQL package procedure DBMS_RESULT_CACHE.MEMORY_REPORT display information to help you determine the amount of memory currently allocated to the result cache. The PL/SQL package function DBMS_RESULT_CACHE.FLUSH clears the result cache and releases all the memory back to the shared pool.

Specifying Miscellaneous SGA Initialization Parameters

- You can set a few additional initialization parameters to control how the SGA uses memory. Physical Memory the LOCK_SGA parameter, when set to TRUE, locks the entire SGA into physical memory. This parameter cannot be used in conjunction with automatic memory management or automatic shared memory management. SGA Starting Address the SHARED_MEMORY_ADDRESS and HI_SHARED_MEMORY_ADDRESS parameters specify the SGA’s starting address at runtime. These parameters are rarely used. For 64-bit platforms, HI_SHARED_MEMORY_ADDRESS specifies the high order 32 bits of the 64-bit address.
- Extended Buffer Cache Mechanism the USE_INDIRECT_DATA_BUFFERS parameter enables the use of the extended buffer cache mechanism for 32-bit platforms that can support more than 4 GB of physical memory. On platforms that do not support this much physical memory, this parameter is ignored. This parameter cannot be used in conjunction with automatic memory management or automatic shared memory management.

3.2. Using Automatic PGA Memory Management

By default, Oracle Database automatically and globally manages the total amount of memory dedicated to the instance PGA. You can control this amount by setting the initialization parameter PGA_AGGREGATE_TARGET. Oracle Database then tries to ensure that the total amount of PGA memory allocated across all database server processes and background processes never exceeds this target. If you create your database with DBCA, you can specify a value for the total instance PGA. DBCA then sets the PGA_AGGREGATE_TARGET initialization parameters in the server parameter file (SPFILE) that it creates. If you do not specify the total instance PGA, DBCA chooses a reasonable default.

If you create the database with the CREATE DATABASE SQL statement and a text initialization parameter file, you can provide a value for PGA_AGGREGATE_TARGET. If you omit this parameter, the database chooses a default value. With automatic PGA memory management, sizing of SQL work areas for all dedicated server sessions is
automatic and all *_AREA_SIZE initialization parameters are ignored for these sessions. At any given time, the total amount of PGA memory available to active work areas on the instance is automatically derived from the parameter PGA_AGGREGATE_TARGET. This amount is set to the value of PGA_AGGREGATE_TARGET minus the PGA memory allocated for other purposes (for example, session memory). The resulting PGA memory is then allotted to individual active work areas based on their specific memory requirements. There are dynamic performance views that provide PGA memory use statistics. Most of these statistics are enabled when PGA_AGGREGATE_TARGET is set.

- Statistics on allocation and use of work area memory can be viewed in the following dynamic performance views:
  - V$SYSSTAT
  - V$SESSTAT
  - V$PGASTAT
  - V$SQL_WORKAREA
  - V$SQL_WORKAREA_ACTIVE

- The following three columns in the V$PROCESS view report the PGA memory allocated and used by an Oracle Database process:
  - PGA_USED_MEM
  - PGA_ALLOCATED_MEM
  - PGA_MAX_MEM

Note: The automatic PGA memory management method applies to work areas allocated by both dedicated and shared server process.

### 3.2.1. Using Manual PGA Memory Management

Oracle Database supports manual PGA memory management, in which you manually tune SQL work areas. In releases earlier than Oracle Database 10g, the database administrator controlled the maximum size of SQL work areas by setting the following parameters: SORT_AREA_SIZE, HASH_AREA_SIZE, BITMAP_MERGE_AREA_SIZE and CREATE_BITMAP_AREA_SIZE. Setting these parameters is difficult, because the maximum work area size is ideally selected from the data input size and the total number of work areas active in the system. These two factors vary greatly from one work area to another and from one time to another. Thus, the various *_AREA_SIZE parameters are difficult to tune under the best of circumstances. For this reason, Oracle strongly recommends that you leave automatic PGA memory management enabled. If you decide to tune SQL work areas manually, you must set the WORKAREA_SIZE_POLICY initialization parameter to MANUAL...

Note: The initialization parameter WORKAREA_SIZE_POLICY is a session- and system-level parameter that can take only two values:

- MANUAL or AUTO. The default is AUTO. You can set PGA_AGGREGATE_TARGET, and then switch back and forth from auto to manual memory management mode. When WORKAREA_SIZE_POLICY is set to AUTO, your settings for *_AREA_SIZE parameters are ignored.

Memory Management Data Dictionary Views

The following dynamic performance views provide information on memory management:

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V$SGA</td>
<td>Displays summary information about the system global area (SGA).</td>
</tr>
<tr>
<td>V$SGAINFO</td>
<td>Displays size information about the SGA, including the sizes of different SGA components, the granule size, and free memory.</td>
</tr>
<tr>
<td>V$SGASTAT</td>
<td>Displays detailed information about how memory is allocated within the shared pool, large pool, Java pool, and Streams pool.</td>
</tr>
<tr>
<td>V$PGASTAT</td>
<td>Displays PGA memory usage statistics as well as statistics about the automatic PGA memory manager when it is enabled</td>
</tr>
</tbody>
</table>
(that is, when PGAAgregarATE_TARGET is set). Cumulative values in V$PGASTAT are accumulated since instance startup.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V$MEMORY_DYNAMIC_COMPONENTS</td>
<td>Displays information on the current size of all automatically tuned and static memory components, with the last operation (for example, grow or shrink) that occurred on each.</td>
</tr>
<tr>
<td>V$SGA_DYNAMIC_COMPONENTS</td>
<td>Displays the current sizes of all SGA components, and the last operation for each component.</td>
</tr>
</tbody>
</table>
4. Users and Resource Management

To connect to the database, each user must specify a valid user name that has been previously defined to the database. An account must have been established for the user, with information about the user being stored in the data dictionary. When you create a database user (account), you specify the following attributes of the user:

- User name
- Authentication method
- Default tablespace
- Temporary tablespace
- Other tablespaces and quotas
- User profile

4.1. Managing User Privileges and Roles

Privileges and roles are used to control user access to data and the types of SQL statements that can be executed. The table that follows describes the three types of privileges and roles:

**System privilege:** system-defined privilege usually granted only by administrators. These privileges allow users to perform specific database operations. **Object privilege:** A system-defined privilege that controls access to a specific object.

**Role:** A collection of privileges and other roles. Some system-defined roles exist, but most are created by administrators. Roles group together privileges and other roles, which facilitates the granting of multiple privileges and roles to users.

Privileges and roles can be granted to other users by users who have been granted the privilege to do so. The granting of roles and privileges starts at the administrator level. At database creation, the administrative user SYS is created and granted all system privileges and predefined Oracle Database roles. User SYS can then grant privileges and roles to other users, and also grant those users the right to grant specific privileges to others.

4.2. Auditing Database Use

You can monitor and record selected user database actions, including those performed by administrators. There are several reasons why you might want to implement database auditing.

**Predefined User Accounts**

Oracle Database includes a number of predefined user accounts. The three types of predefined accounts are:

- Administrative accounts (SYS, SYSTEM, SYSMAN, and DBSNMP)
- The management agent of Enterprise Manager uses the DBSNMP account to monitor and manage the database. You must not delete these accounts.
- Sample schema accounts
- These accounts are used for examples in Oracle Database documentation and instructional materials. Examples are HR, SH, and OE. You must unlock these accounts and reset their passwords before using them.

4.3. Creating a New User Account

You create a database user with the CREATE USER statement. To create a user, you must have the CREATE USER system privilege. Because it is a powerful privilege, a database administrator or security administrator is usually the only user who has the CREATE USER system privilege. Example creates a user and specifies the user...
password, default tablespace, temporary tablespace where temporary segments are created, tablespace quotas, and profile. It also grants the user the minimum privileges, CONNECT and CREATE SESSION, to log in to the database session.

**Example:** Creating a User Account with CONNECT and CREATE SESSION Privileges

```
CREATE USER jward
  IDENTIFIED BY password
  DEFAULT TABLESPACE data_ts
  QUOTA 100M ON test_ts
  QUOTA 500K ON data_ts
  TEMPORARY TABLESPACE temp_ts
  PROFILE clerk;
GRANT CREATE SESSION TO jward;
```

A newly created user cannot connect to the database until you grant the user the CREATE SESSION system privileges. So, immediately after you create the user account, use the GRANT SQL statement to grant the user these privileges. If the user needs to access Oracle Enterprise Manager, you should also grant the user the SELECT ANY DICTIONARY privilege.

**Note:** As a security administrator, you should create your own roles and assign only those privileges that are needed. For example, many users formerly granted the CONNECT privilege did not need the additional privileges CONNECT used to provide. Instead, only CREATE SESSION was actually needed, and in fact, that is the only privilege CONNECT presently retains. Creating organization-specific roles gives an organization detailed control of the privileges it assigns, and protects it in case Oracle Database changes the roles that it defines in future releases. For example, both CONNECT and RESOURCE roles will be deprecated in future Oracle Database releases.

**Specifying a User Name**

Within each database, a user name must be unique with respect to other user names and roles. A user and role cannot have the same name. Furthermore, each user has an associated schema. Within a schema, each schema object must have a unique name.

```
CREATE USER jward
  IDENTIFIED BY password
  DEFAULT TABLESPACE data_ts
  QUOTA 100M ON test_ts
  QUOTA 500K ON data_ts
  TEMPORARY TABLESPACE temp_ts
  PROFILE clerk;
```

**Assigning the User a Password**

In this, the new user is to be authenticated using the database. In this case, the connecting user must supply the correct password to the database to connect successfully. To specify a password for the user, use the IDENTIFIED BY clause in the CREATE USER statement.

```
CREATE USER jward
  IDENTIFIED BY password
  DEFAULT TABLESPACE data_ts
  QUOTA 100M ON test_ts
  QUOTA 500K ON data_ts
  TEMPORARY TABLESPACE temp_ts
  PROFILE clerk;
```

### 4.4. Assigning a Default Tablespace for the User

Each user should have a default tablespace. When a user creates a schema object and specifies no tablespace to contain it, Oracle Database stores the object in the default user tablespace. The default setting for the default tablespaces of all users is the SYSTEM tablespace. If a user does not create objects, and has no privileges to do
so, then this default setting is fine. However, if a user is likely to create any type of object, then you should specifically assign the user a default tablespace, such as the USERS tablespace. Using a tablespace other than SYSTEM reduces contention between data dictionary objects and user objects for the same data files. In general, do not store user data in the SYSTEM tablespace.

You can use the CREATE TABLESPACE SQL statement to create a permanent default tablespace other than SYSTEM at the time of database creation, to be used as the database default for permanent objects. By separating the user data from the system data, you reduce the likelihood of problems with the SYSTEM tablespace, which can in some circumstances cause the entire database to become nonfunctional. This default permanent tablespace is not used by system users, that is, SYS, SYSTEM, and OUTLN, whose default permanent tablespace is SYSTEM. A tablespace designated as the default permanent tablespace cannot be dropped. To accomplish this goal, you must first designate another tablespace as the default permanent tablespace. You can use the ALTER TABLESPACE SQL statement to alter the default permanent tablespace to another tablespace. Be aware that this will affect all users or objects created after the ALTER DDL statement commits.

You can also set a user default tablespace during user creation, and change it later with the ALTER USER statement. Changing the user default tablespace affects only objects created after the setting is changed. When you specify the default tablespace for a user, also specify a quota on that tablespace.

In the following CREATE USER statement, the default tablespace for user jward is data_ts, and his quota on that tablespace is 500K:

```sql
CREATE USER jward
  IDENTIFIED BY password
  DEFAULT TABLESPACE data_ts
  QUOTA 100M ON test_ts
  QUOTA 500K ON data_ts
  TEMPORARY TABLESPACE temp_ts
  PROFILE clerk;
```

### 4.5. Assigning a Tablespace Quota for the User

You can assign each user a tablespace quota for any tablespace (except a temporary tablespace). Assigning a quota accomplishes the following:

- Users with privileges to create certain types of objects can create those objects in the specified tablespace.
- Oracle Database limits the amount of space that can be allocated for storage of a user's objects within the specified tablespace to the amount of the quota.
- By default, a user has no quota on any tablespace in the database. If the user has the privilege to create a schema object, then you must assign a quota to allow the user to create objects. At a minimum, assign users a quota for the default tablespace, and additional quotas for other tablespaces in which they can create objects.

The following CREATE USER statement assigns the following quotas for the test_ts and data_ts tablespaces:

```sql
CREATE USER jward
  IDENTIFIED BY password
  DEFAULT TABLESPACE data_ts
  QUOTA 100M ON test_ts
  QUOTA 500K ON data_ts
  TEMPORARY TABLESPACE temp_ts
  PROFILE clerk;
```

You can assign a user either individual quotas for a specific amount of disk space in each tablespace or an unlimited amount of disk space in all tablespaces. Specific quotas prevent a user's objects from using too much
space in the database. You can assign quotas to a user tablespace when you create the user, or add or change quotas later. (You can find existing user quotas by querying the USER_TS QUOTAS view.) If a new quota is less than the old one, then the following conditions remain true:

- If a user has already exceeded a new tablespace quota, then the objects of a user in the tablespace cannot be allocated more space until the combined space of these objects is less than the new quota.
- If a user has not exceeded a new tablespace quota, or if the space used by the objects of the user in the tablespace falls under a new tablespace quota, then the user's objects can be allocated space up to the new quota.

4.6 Revoking the Ability for Users to Create Objects in a Tablespace

You can revoke the ability of a user to create objects in a tablespace by using the ALTER USER SQL statement to change the current quota of the user to zero. After a quota of zero is assigned, the objects of the user in the tablespace remain, but the user cannot create new objects, nor can existing objects be allocated any new space.

4.6.1 Granting Users the UNLIMITED TABLESPACE System Privilege

To permit a user to use an unlimited amount of any tablespace in the database, grant the user the UNLIMITED TABLESPACE system privilege. This overrides all explicit tablespace quotas for the user. If you later revoke the privilege, then explicit quotas again take effect. You can grant this privilege only to users, not to roles. Before granting the UNLIMITED TABLESPACE system privilege, you must consider the consequences of doing so.

Advantage:
- You can grant a user unlimited access to all tablespaces of a database with one statement.

Disadvantages:
- The privilege overrides all explicit tablespace quotas for the user.
- You cannot selectively revoke tablespace access from a user with the UNLIMITED TABLESPACE privilege. You can grant selective or restricted access only after revoking the privilege.

4.7. Assigning a Temporary Tablespace for the User

You should assign each user a temporary tablespace. When a user executes a SQL statement that requires a temporary segment, Oracle Database stores the segment in the temporary tablespace of the user. These temporary segments are created by the system when performing sort or join operations. Temporary segments are owned by SYS, which has resource privileges in all tablespaces. In the following, the temporary tablespace of jward is temp_ts, a tablespace created explicitly to contain only temporary segments.

```
CREATE USER jward
    IDENTIFIED BY password
    DEFAULT TABLESPACE data_ts
    QUOTA 100M ON test_ts
    QUOTA 500K ON data_ts
    TEMPORARY TABLESPACE temp_ts
    PROFILE clerk;
```

To create a temporary tablespace, use the CREATE TEMPORARY TABLESPACE SQL statement. If you do not explicitly assign the user a temporary tablespace, then Oracle Database assigns the user the default temporary tablespace that was specified at database creation or by an ALTER DATABASE statement at a later time. If there is no default temporary tablespace explicitly assigned, then the default is the SYSTEM tablespace or another permanent default established by the system administrator. Do not store user data in the SYSTEM tablespace. Assigning a tablespace to be used specifically as a temporary tablespace eliminates file contention among temporary segments and other types of segments.
Note: If your SYSTEM tablespace is locally managed, then users must be assigned a specific default (locally managed) temporary tablespace. They may not be allowed to default to using the SYSTEM tablespace because temporary objects cannot be placed in locally managed permanent tablespaces.

You can set the temporary tablespace for a user at user creation, and change it later using the ALTER USER statement. If you are logged in as user SYS, you can set a quota for the temporary tablespace, as well as other space allocations. (Only user SYS can do this, because all space in the temporary tablespace belongs to user SYS.) You can also establish tablespace groups instead of assigning individual temporary tablespaces.

4.8. Specifying a Profile for the User

You can specify a profile when you create a user. A profile is a set of limits on database resources and password access to the database. If you do not specify a profile, then Oracle Database assigns the user a default profile. The following example demonstrates how to assign a user a profile.

```sql
CREATE USER jward
  IDENTIFIED BY password
  DEFAULT TABLESPACE data_ts
  QUOTA 100M ON test_ts
  QUOTA 500K ON data_ts
  TEMPORARY TABLESPACE temp_ts
  PROFILE clerk;
```

4.9 Setting a Default Role for the User

A role is a named group of related privileges that you grant as a group to users or other roles. A default role is automatically enabled for a user when the user creates a session. You can assign a user zero or more default roles. You cannot set default roles for a user in the CREATE USER statement. When you first create a user, the default role setting for the user is ALL, which causes all roles subsequently granted to the user to be default roles. Use the ALTER USER statement to change the default roles for the user. For example:

```sql
GRANT USER jward clerk_role;
ALTER USER jward DEFAULT ROLE clerk_role;
```

Before a role can be made the default role for a user, that user needs to have been already granted the role.

4.10. Altering User Accounts

Users can change their own passwords. However, to change any other option of a user security domain, you must have the ALTER USER system privilege. Security administrators are typically the only users that have this system privilege, as it allows a modification of any user security domain. This privilege includes the ability to set tablespace quotas for a user on any tablespace in the database, even if the user performing the modification does not have a quota for a specified tablespace. You can alter user security settings with the ALTER USER SQL statement. Changing user security settings affects the future user sessions, not current sessions. Example shows how to use the ALTER USER statement to alter the security settings for the user avyrros:

**Example:** Altering a User Account

```sql
ALTER USER avyrros
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE data_ts
  TEMPORARY TABLESPACE temp_ts
  QUOTA 100M ON data_ts
  QUOTA 0 ON test_ts
  PROFILE clerk;
```

The ALTER USER statement here changes the security settings for the user avyrros as follows:
• Authentication is changed to use the operating system account of the user avyrros.
• The default and temporary tablespaces are explicitly set for user AVYRROS.
• The user avyrros is given a 100M quota for the DATA_TS tablespace.
• The quota on the test_ts is revoked for the user avyrros.
• The user avyrros is assigned the clerk profile.

4.10.1. Changing the User Password
Most users can change their own passwords with the PASSWORD statement, as follows:

```
PASSWORD andy
Changing password for andy
New password: password
Retype new password: password
```

No special privileges (other than those to connect to the database and create a session) are required for a user to change his or her own password. Encourage users to change their passwords frequently. Users can also use the ALTER USER SQL statement to change their passwords. For example:

```
ALTER USER andy IDENTIFIED BY password;
```

However, for better security, use the PASSWORD statement to change the account's password. The ALTER USER statement displays the new password on the screen, where it can be seen by any overly curious coworkers. The PASSWORD command does not display the new password, so it is only known to you, not to your co-workers. In both cases, the password is encrypted on the network. Users must have the PASSWORD and ALTER USER privilege to switch between methods of authentication. Usually, only an administrator has this privilege.

4.11. Types of System Resources and Limits
Oracle Database can limit the use of several types of system resources, including CPU time and logical reads. In general, you can control each of these resources at the session level, call level, or both, as discussed in the following sections:

4.11.1. Limiting the User Session Level
Each time a user connects to a database, a session is created. Each session uses CPU time and memory on the computer that runs Oracle Database. You can set several resource limits at the session level.

If a user exceeds a session-level resource limit, then Oracle Database terminates (rolls back) the current statement and returns a message indicating that the session limit has been reached. At this point, all previous statements in the current transaction are intact, and the only operations the user can perform are COMMIT, ROLLBACK, or disconnect (in this case, the current transaction is committed). All other operations produce an error. Even after the transaction is committed or rolled back, the user cannot accomplish any more work during the current session.

4.11.2. Limiting Database Call Levels
Each time a user runs a SQL statement, Oracle Database performs several steps to process the statement. During this processing, several calls are made to the database as a part of the different execution phases. To prevent any one call from using the system excessively, Oracle Database lets you set several resource limits at the call level. If a user exceeds a call-level resource limit, then Oracle Database halts the processing of the statement, rolls back the statement, and returns an error. However, all previous statements of the current transaction remain intact, and the user session remains connected.

4.11.3. Limiting CPU Time
When SQL statements and other types of calls are made to Oracle Database, a certain amount of CPU time is necessary to process the call. Average calls require a small amount of CPU time. However, a SQL statement involving a large amount of data or a runaway query can potentially use a large amount of CPU time, reducing CPU
time available for other processing. To prevent uncontrolled use of CPU time, you can set fixed or dynamic limits on the CPU time for each call and the total amount of CPU time used for Oracle Database calls during a session. The limits are set and measured in CPU one-hundredth seconds (0.01 seconds) used by a call or a session.

4.11.4. Limiting Logical Reads
Input/output (I/O) is one of the most expensive operations in a database system. SQL statements that are I/O-intensive can monopolize memory and disk use and cause other database operations to compete for these resources. To prevent single sources of excessive I/O, you can limit the logical data block reads for each call and for each session. Logical data block reads include data block reads from both memory and disk. The limits are set and measured in number of block reads performed by a call or during a session.

4.11.5. Limiting Other Resources
Oracle Database provides for limiting several other resources at the session level:

- You can limit the number of concurrent sessions for each user. Each user can create only up to a predefined number of concurrent sessions.
- You can limit the idle time for a session. If the time between calls in a session reaches the idle time limit, then the current transaction is rolled back, the session is terminated, and the resources of the session are returned to the system. The next call receives an error that indicates that the user is no longer connected to the instance. This limit is set as a number of elapsed minutes.

Note: Shortly after a session is terminated because it has exceeded an idle time limit, the process monitor (PMON) background process cleans up after the terminated session. Until PMON completes this process, the terminated session is still counted in any session or user resource limit.

- You can limit the elapsed connect time for each session. If the duration of a session exceeds the elapsed time limit, then the current transaction is rolled back, the session is dropped, and the resources of the session are returned to the system. This limit is set as a number of elapsed minutes.

Note: Oracle Database does not constantly monitor the elapsed idle time or elapsed connection time. Doing so reduces system performance. Instead, it checks every few minutes. Therefore, a session can exceed this limit slightly (for example, by 5 minutes) before Oracle Database enforces the limit and terminates the session.

You can limit the amount of private System Global Area (SGA) space (used for private SQL areas) for a session. This limit is only important in systems that use the shared server configuration. Otherwise, private SQL areas are located in the Program Global Area (PGA). This limit is set as a number of bytes of memory in the SGA of an instance. Use the characters K or M to specify kilobytes or megabytes.

4.11.6. Determining Values for Resource Limits of Profiles
Before creating profiles and setting the resource limits associated with them, you should determine appropriate values for each resource limit. You can base these values on the type of operations a typical user performs. For example, if one class of user does not usually perform a high number of logical data block reads, then use the ALTER RESOURCE COST SQL statement to set the LOGICAL_READS_PER_SESSION setting conservatively. Usually, the best way to determine the appropriate resource limit values for a given user profile is to gather historical information about each type of resource usage. For example, the database or security administrator can use the AUDIT SESSION clause to gather information about the limits CONNECT_TIME, LOGICAL_READS_PER_SESSION. You can gather statistics for other limits using the Monitor feature of Oracle Enterprise Manager (or SQL*Plus), specifically the Statistics monitor.

4.12 Managing Resources with Profiles
A profile is a named set of resource limits and password parameters that restrict database usage and instance resources for a user. You can assign a profile to each user, and a default profile to all others. Each user can have only one profile, and creating a new one supersedes an earlier version. You need to create and manage user profiles only if resource limits are a requirement of your database security policy. To use profiles, first categorize the
related types of users in a database. Just as roles are used to manage the privileges of related users, profiles are used to manage the resource limits of related users. Determine how many profiles are needed to encompass all types of users in a database and then determine appropriate resource limits for each profile.

In general, the word profile refers to a collection of attributes that apply to a user, enabling a single point of reference for any of multiple users that share those exact attributes. User profiles in Oracle Internet Directory contain attributes pertinent to directory usage and authentication for each user. Similarly, profiles in Oracle Label Security contain attributes useful in label security user administration and operations management. Profile attributes can include restrictions on system resources. You can use Database Resource Manager to set these types of resource limits. Profile resource limits are enforced only when you enable resource limitation for the associated database. Enabling this limitation can occur either before starting up the database (using the RESOURCE_LIMIT initialization parameter) or while it is open (using the ALTER SYSTEM statement). Thought password parameters reside in profiles, they are unaffected by RESOURCE_LIMIT or ALTER SYSTEM and password management is always enabled. In Oracle Database, Database Resource Manager primarily handles resource allocations and restrictions.

4.12.1. Creating Profiles

Any authorized database user can create, assign to users, alter, and drop a profile at any time (using the CREATE USER or ALTER USER statement). Profiles can be assigned only to users and not to roles or other profiles. Profile assignments do not affect current sessions, instead, they take effect only in subsequent sessions. To find information about current profiles, query the DBA_PROFILES view.

4.12.2. Dropping Profiles

To drop a profile, you must have the DROP PROFILE system privilege. You can drop a profile (other than the default profile) using the SQL statement DROP PROFILE. To successfully drop a profile currently assigned to a user, use the CASCADE option.

The following statement drops the profile clerk, even though it is assigned to a user:

```
DROP PROFILE clerk CASCADE;
```

Any user currently assigned to a profile that is dropped is automatically assigned to the DEFAULT profile. The DEFAULT profile cannot be dropped. When a profile is dropped, the drop does not affect currently active sessions. Only sessions created after a profile is dropped use the modified profile assignments.

4.12.3 Deleting User Accounts

When you drop a user account, Oracle Database removes the user account and associated schema from the data dictionary. It also immediately drops all schema objects contained in the user schema, if any.

Note:

- If a user schema and associated objects must remain but the user must be denied access to the database, then revoke the CREATE SESSION privilege from the user.
- Do not attempt to drop the SYS or SYSTEM user. Doing so corrupts your database.
- A user that is currently connected to a database cannot be dropped. To drop a connected user, you must first terminate the user sessions using the SQL statement ALTER SYSTEM with the KILL SESSION clause. You can find the session ID (SID) by querying the V$SESSION view.

Example shows how to query V$SESSION and displays the session ID, serial number, and user name for user ANDY.

3 Querying V$SESSION for the Session ID of a User

```
SELECT SID, serial#, username FROM V$SESSION;
```

<table>
<thead>
<tr>
<th>SID</th>
<th>SERIAL#</th>
<th>USERNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>---------</td>
<td>----------</td>
</tr>
</tbody>
</table>

www.wilshiresoft.com                         Wilshire Software Technologies        Rev. Dt: 10-Oct-08
info@wilshiresoft.com                       Ph: 2761-2214 / 5577-2214   Version: 6
127 55234 ANDY

Example shows how to kill the session for user ANDY.

4.12.4. Killing a User Session

ALTER SYSTEM KILL SESSION '127, 55234';

You can drop a user from a database using the DROP USER statement. To drop a user and all the user schema objects (if any), you must have the DROP USER system privilege. Because the DROP USER system privilege is powerful, a security administrator is typically the only type of user that has this privilege. If the schema of the user contains any dependent schema objects, then use the CASCADE option to drop the user and all associated objects and foreign keys that depend on the tables of the user successfully. If you do not specify CASCADE and the user schema contains dependent objects, then an error message is returned and the user is not dropped.

Before dropping a user whose schema contains objects, thoroughly investigate which objects the schema contains and the implications of dropping them. You can find the objects owned by a particular user by querying the DBA_OBJECTS view. Finding Objects Owned by a User

SELECT OWNER, OBJECT_NAME FROM DBA_OBJECTS WHERE OWNER LIKE 'ANDY';

(Enter the user name in capital letters.) Pay attention to any unknown cascading effects. For example, if you intend to drop a user who owns a table, then check whether any views or procedures depend on that particular table. Example drops the user andy and all associated objects and foreign keys that depend on the tables owned by andy.

4.12.5. Dropping a User Account

DROP USER andy CASCADE;

Table lists data dictionary views that contain information about database users and profiles.

Data Dictionary Views That Contain User and Profile Information

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL_OBJECTS</td>
<td>Describes all objects accessible to the current user</td>
</tr>
<tr>
<td>ALL_USERS</td>
<td>Lists users visible to the current user, but does not describe them</td>
</tr>
<tr>
<td>DBA_PROFILES</td>
<td>Displays all profiles and their limits</td>
</tr>
<tr>
<td>DBA_TS_QUOTAS</td>
<td>Describes tablespace quotas for users</td>
</tr>
<tr>
<td>DBA_OBJECTS</td>
<td>Describes all objects in the database</td>
</tr>
<tr>
<td>DBA_USERS</td>
<td>Describes all users of the database</td>
</tr>
<tr>
<td>DBA_USERS_WITH_DEFPWD</td>
<td>Lists all user accounts that have default passwords</td>
</tr>
<tr>
<td>PROXY_USERS</td>
<td>Describes users who can assume the identity of other users</td>
</tr>
<tr>
<td>RESOURCE_COST</td>
<td>Lists the cost for each resource in terms of CPUs for each session, reads for each session, connection times, and SGA</td>
</tr>
<tr>
<td>USER_PASSWORD_LIMITS</td>
<td>Describes the password profile parameters that are assigned to the user</td>
</tr>
<tr>
<td>USER_RESOURCE_LIMITS</td>
<td>Displays the resource limits for the current user</td>
</tr>
<tr>
<td>USER_TS_QUOTAS</td>
<td>Describes tablespace quotas for users</td>
</tr>
<tr>
<td>USER_OBJECTS</td>
<td>Describes all objects owned by the current user</td>
</tr>
<tr>
<td>USER_USERS</td>
<td>Describes only the current user</td>
</tr>
<tr>
<td>V$SESSION</td>
<td>Lists session information for each current session, includes user name</td>
</tr>
<tr>
<td>V$SESSTAT</td>
<td>Lists user session statistics</td>
</tr>
</tbody>
</table>
The following sections present examples of using these views. These examples assume that the following the following statements have been run:

```
CREATE PROFILE clerk LIMIT
    SESSIONS_PER_USER 1
    IDLE_TIME 30
    CONNECT_TIME 600;
CREATE USER jfee
    IDENTIFIED BY password
    DEFAULT TABLESPACE users
    TEMPORARY TABLESPACE temp_ts
    QUOTA 500K ON users
    PROFILE clerk;
CREATE USER dcranney
    IDENTIFIED BY password
    DEFAULT TABLESPACE users
    TEMPORARY TABLESPACE temp_ts
    QUOTA unlimited ON users;
CREATE USER userscott
    IDENTIFIED BY password;
```

### 4.13 Listing All Users and Associated Information

To find all users and their associated information as defined in the database, query the DBA_USERS view. For example:

```
SELECT USERNAME, PROFILE, ACCOUNT_STATUS FROM DBA_USERS;
```

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>PROFILE</th>
<th>ACCOUNT_STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>DEFAULT</td>
<td>OPEN</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>DEFAULT</td>
<td>OPEN</td>
</tr>
<tr>
<td>USERScott</td>
<td>DEFAULT</td>
<td>OPEN</td>
</tr>
<tr>
<td>JFEE</td>
<td>CLERK</td>
<td>OPEN</td>
</tr>
<tr>
<td>DCRANNEY</td>
<td>DEFAULT</td>
<td>OPEN</td>
</tr>
</tbody>
</table>

#### 4.13.1. Listing All Tablespace Quotas

Use the DBA_TS_QUOTAS view to list all tablespace quotas specifically assigned to each user. For example:

```
SELECT * FROM DBA_TS_QUOTAS;
```

<table>
<thead>
<tr>
<th>TABLESPACE</th>
<th>USERNAME</th>
<th>BYTES</th>
<th>MAX_BYTES</th>
<th>BLOCKS</th>
<th>MAX_BLOCKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERS</td>
<td>JFEE</td>
<td>0</td>
<td>512000</td>
<td>0</td>
<td>250</td>
</tr>
<tr>
<td>USERS</td>
<td>DCRANNEY</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

When specific quotas are assigned, the exact number is indicated in the MAX_BYTES column. This number is always a multiple of the database block size, so if you specify a tablespace quota that is not a multiple of the database block size, then it is rounded up accordingly. Unlimited quotas are indicated by -1.

#### 4.13.2. Listing All Profiles and Assigned Limits

The DBA_PROFILE view lists all profiles in the database and associated settings for each limit in each profile. For
4.14 Viewing Memory Use for Each User Session

To find the memory use for each user session, query the V$SESSION view. The following query lists all current sessions, showing the Oracle Database user and current User Global Area (UGA) memory use for each session:

```sql
SELECT USERNAME, VALUE || 'bytes' "Current UGA memory"
FROM V$SESSION sess, V$SESSTAT stat, V$STATNAME name
WHERE sess.SID = stat.SID
AND stat.STATISTIC# = name.STATISTIC#
AND name.NAME = 'session uga memory';
```

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>Current UGA memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1863bytes</td>
<td></td>
</tr>
<tr>
<td>17464bytes</td>
<td></td>
</tr>
<tr>
<td>19180bytes</td>
<td></td>
</tr>
<tr>
<td>18364bytes</td>
<td></td>
</tr>
<tr>
<td>39384bytes</td>
<td></td>
</tr>
<tr>
<td>35292bytes</td>
<td></td>
</tr>
</tbody>
</table>

32 rows selected.
17696 bytes
15868 bytes

USERSCOTT 42244 bytes
SYS 98196 bytes
SYSTEM 30648 bytes

11 rows selected.
5. Controlfile Management

5.1. What Is a Control File?

Every Oracle Database has a control file, which is a small binary file that records the physical structure of the database. The control file includes:

- The database name
- Names and locations of associated datafiles and redo log files
- The timestamp of the database creation
- The current log sequence number
- Checkpoint information

The control file must be available for writing by the Oracle Database server whenever the database is open. Without the control file, the database cannot be mounted and recovery is difficult. The control file of an Oracle Database is created at the same time as the database. By default, at least one copy of the control file is created during database creation. On some operating systems the default is to create multiple copies. You should create two or more copies of the control file during database creation. You can also create control files later, if you lose control files or want to change particular settings in the control files.

5.2. Guidelines for Control Files

This section describes guidelines you can use to manage the control files for a database, and contains the following topics:

- Provide Filenames for the Control Files
- Multiplex Control Files on Different Disks
- Back Up Control Files
- Manage the Size of Control Files

5.2.1. Provide Filenames for the Control Files

You specify control file names using the CONTROL_FILES initialization parameter in the database initialization parameter file. The instance recognizes and opens all the listed file during startup and the instance writes to and maintains all listed control files during database operation. If you do not specify files for CONTROL_FILES before database creation:

- If you are not using Oracle-managed files, then the database creates a control file and uses a default filename. The default name is operating system specific.
- If you are using Oracle-managed files, then the initialization parameters you set to enable that feature determine the name and location of the control files.
- If you are using Automatic Storage Management, you can place incomplete ASM filenames in the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST initialization parameters. ASM then automatically creates control files in the appropriate places.

5.2.2. Multiplex Control Files on Different Disks

Every Oracle Database should have at least two control files, each stored on a different physical disk. If a control file is damaged due to a disk failure, the associated instance must be shut down. Once the disk drive is repaired, the damaged control file can be restored using the intact copy of the control file from the other disk and the instance can be restarted. In this case, no media recovery is required. The behavior of multiplexed control files is this:
• The database writes to all filenames listed for the initialization parameter CONTROL_FILES in the database initialization parameter file.
• The database reads only the first file listed in the CONTROL_FILES parameter during database operation.
• If any of the control files become unavailable during database operation, the instance becomes inoperable and should be aborted.

Note: Oracle strongly recommends that your database has a minimum of two control files and that they are located on separate physical disks. One way to multiplex control files is to store a control file copy on every disk drive that stores members of redo log groups, if the redo log is multiplexed. By storing control files in these locations, you minimize the risk that all control files and all groups of the redo log will be lost in a single disk failure.

5.2.3. Back Up Control Files
It is very important that you back up your control files. This is true initially, and every time you change the physical structure of your database. Such structural changes include:

• Adding, dropping, or renaming datafiles
• Adding or dropping a tablespace, or altering the read/write state of the tablespace
• Adding or dropping redo log files or groups

5.2.4. Manage the Size of Control Files
The main determinants of the size of a control file are the values set for the MAXDATAFILES, MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, and MAXINSTANCES parameters in the CREATE DATABASE statement that created the associated database. Increasing the values of these parameters increases the size of a control file of the associated database.

5.3. Creating Control Files
Creating Initial Control Files
The initial control files of an Oracle Database are created when you issue the CREATE DATABASE statement. The names of the control files are specified by the CONTROL_FILES parameter in the initialization parameter file used during database creation. The filenames specified in CONTROL_FILES should be fully specified and are operating system specific. The following is an example of a CONTROL_FILES initialization parameter:

CONTROL_FILES = (/u01/oracle/prod/control01.ctl, /u02/oracle/prod/control02.ctl, /u03/oracle/prod/control03.ctl)

If files with the specified names currently exist at the time of database creation, you must specify the CONTROLFILE REUSE clause in the CREATE DATABASE statement, or else an error occurs. Also, if the size of the old control file differs from the SIZE parameter of the new one, you cannot use the REUSE clause. The size of the control file changes between some releases of Oracle Database, as well as when the number of files specified in the control files changes. Configuration parameters such as MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, MAXDATAFILES, and MAXINSTANCES affect control file size. You can subsequently change the value of the CONTROL_FILES initialization parameter to add more control files or to change the names or locations of existing control files.

5.3.1. Creating Additional Copies, Renaming, and Relocating Control Files
You can create an additional control file copy for multiplexing by copying an existing control file to a new location and adding the file name to the list of control files. Similarly, you rename an existing control file by copying the file to its new name or location, and changing the file name in the control file list. In both cases, to guarantee that control files do not change during the procedure, shut down the database before copying the control file.

To add a multiplexed copy of the current control file or to rename a control file:
1. Shut down the database.
2. Copy an existing control file to a new location, using operating system commands.
3. Edit the CONTROL_FILES parameter in the database initialization parameter file to add the new control file name, or to change the existing control filename.
4. Restart the database.

5.3.2. Creating New Control Files

This section discusses when and how to create new control files.

When to Create New Control Files

It is necessary for you to create new control files in the following situations:

- All control files for the database have been permanently damaged and you do not have a control file backup.
- You want to change the database name.

For example, you would change a database name if it conflicted with another database name in a distributed environment.

**Note:** You can change the database name and DBID (internal database identifier) using the DBNEWID utility.

The compatibility level is set to a value that is earlier than 10.2.0, and you must make a change to an area of database configuration that relates to any of the following parameters from the CREATE DATABASE or CREATE CONTROLFILE commands: MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, and MAXINSTANCES. If compatibility is 10.2.0 or later, you do not have to create new control files when you make such a change; the control files automatically expand, if necessary, to accommodate the new configuration information. For example, assume that when you created the database or recreated the control files, you set MAXLOGFILES to 3. Suppose that now you want to add a fourth redo log file group to the database with the ALTER DATABASE command. If compatibility is set to 10.2.0 or later, you can do so and the controlfiles automatically expand to accommodate the new logfile information. However, with compatibility set earlier than 10.2.0, your ALTER DATABASE command would generate an error, and you would have to first create new control files.

The CREATE CONTROLFILE Statement

You can create a new control file for a database using the CREATE CONTROLFILE statement. The following statement creates a new control file for the prod database (a database that formerly used a different database name):

```
CREATE CONTROLFILE
SET DATABASE prod
LOGFILE GROUP 1 ('/u01/oracle/prod/redo01_01.log',
                 '/u01/oracle/prod/redo01_02.log'),
               GROUP 2 ('/u01/oracle/prod/redo02_01.log',
                 '/u01/oracle/prod/redo02_02.log'),
               GROUP 3 ('/u01/oracle/prod/redo03_01.log',
                 '/u01/oracle/prod/redo03_02.log')
RESETLOGS
DATAFILE '/u01/oracle/prod/system01.dbf' SIZE 3M,
          '/u01/oracle/prod/rbs01.dbs' SIZE 5M,
          '/u01/oracle/prod/users01.dbs' SIZE 5M,
          '/u01/oracle/prod/temp01.dbs' SIZE 5M
MAXLOGFILES 50
MAXLOGMEMBERS 3
MAXLOGHISTORY 400
MAXDATAFILES 200
MAXINSTANCES 6
ARCHIVELOG;
```

Cautions: The CREATE CONTROLFILE statement can potentially damage specified datafiles and redo log files.
Omitting a filename can cause loss of the data in that file, or loss of access to the entire database. If the database had forced logging enabled before creating the new control file, and you want it to continue to be enabled, then you must specify the FORCE LOGGING clause in the CREATE CONTROLFILE statement.

5.3.3. Steps for Creating New Control Files

Complete the following steps to create a new control file.

Make a list of all datafiles and redo log files of the database. You will already have a list of datafiles and redo log files that reflect the current structure of the database. However, if you have no such list, executing the following statements will produce one.

```
SELECT MEMBER FROM V$LOGFILE;
SELECT NAME FROM V$DATAFILE;
SELECT VALUE FROM V$PARAMETER WHERE NAME = 'control_files';
```

1. If you have no such lists and your control file has been damaged so that the database cannot be opened, try to locate all of the datafiles and redo log files that constitute the database. Any files not specified in step 5 are not recoverable once a new control file has been created. Moreover, if you omit any of the files that make up the SYSTEM tablespace, you might not be able to recover the database.

2. Shut down the database.

3. Shut down the database normally if possible. Use the IMMEDIATE or ABORT clauses only as a last resort.

4. Start up a new instance, but do not mount or open the database:

5. Create a new control file for the database using the CREATE CONTROLFILE statement.

When creating a new control file, specify the RESETLOGS clause if you have lost any redo log groups in addition to control files. In this case, you will need to recover from the loss of the redo logs (step 8). You must specify the RESETLOGS clause if you have renamed the database. Otherwise, select the NORESETLOGS clause.

6. Store a backup of the new control file on an offline storage device.

7. Edit the CONTROL_FILES initialization parameter for the database to indicate all of the control files now part of your database as created in step 5 (not including the backup control file). If you are renaming the database, edit the DB_NAME parameter in your instance parameter file to specify the new name.

8. Recover the database if necessary. If you are not recovering the database, skip to step 9.

9. Open the database using one of the following methods:

   - If you did not perform recovery, or you performed complete, closed database recovery in step 8, open the database normally.
   - ALTER DATABASE OPEN;

   - If you specified RESETLOGS when creating the control file, use the ALTER DATABASE statement, indicating RESETLOGS.
   - ALTER DATABASE OPEN RESETLOGS;

   The database is now open and available for use.

5.4. Recovering a Control File Using a Current Copy

This section presents ways that you can recover your control file from a current backup or from a multiplexed copy.

5.4.1. Recovering from Control File Corruption Using a Control File Copy

This procedure assumes that one of the control files specified in the CONTROL_FILES parameter is corrupted, that the control file directory is still accessible, and that you have a multiplexed copy of the control file.

1. With the instance shut down, use an operating system command to overwrite the bad control file with a good
copy:
    % cp /u03/oracle/prod/control03.ctl /u02/oracle/prod/control02.ctl

2. Start SQL*Plus and open the database:
   SQL> STARTUP

5.4.2. Recovering from Permanent Media Failure Using a Control File Copy

This procedure assumes that one of the control files specified in the CONTROL_FILES parameter is inaccessible due to a permanent media failure and that you have a multiplexed copy of the control file.

1. With the instance shut down, use an operating system command to copy the current copy of the control file to a new, accessible location:
    % cp /u01/oracle/prod/control01.ctl /u04/oracle/prod/control03.ctl

2. Edit the CONTROL_FILES parameter in the initialization parameter file to replace the bad location with the new location:
   CONTROL_FILES = (/u01/oracle/prod/control01.ctl,
                      /u02/oracle/prod/control02.ctl, /u04/oracle/prod/control03.ctl)

3. Start SQL*Plus and open the database:
   SQL> STARTUP

If you have multiplexed control files, you can get the database started up quickly by editing the CONTROL_FILES initialization parameter. Remove the bad control file from CONTROL_FILES setting and you can restart the database immediately. Then you can perform the reconstruction of the bad control file and at some later time shut down and restart the database after editing the CONTROL_FILES initialization parameter to include the recovered control file.

5.5. Dropping Control Files

You want to drop control files from the database, for example, if the location of a control file is no longer appropriate. Remember that the database should have at least two control files at all times.

1. Shut down the database.
2. Edit the CONTROL_FILES parameter in the database initialization parameter file to delete the old control file name.
3. Restart the database.

Note: This operation does not physically delete the unwanted control file from the disk. Use operating system commands to delete the unnecessary file after you have dropped the control file from the database.

Control Files Data Dictionary Views
The following views display information about control files:

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V$DATABASE</td>
<td>Displays database information from the control file</td>
</tr>
<tr>
<td>V$CONTROLFILE</td>
<td>Lists the names of control files</td>
</tr>
<tr>
<td>V$CONTROLFILE_RECORD_SECTION</td>
<td>Displays information about control file record sections</td>
</tr>
<tr>
<td>V$PARAMETER</td>
<td>Displays the names of control files as specified in the CONTROL_FILES initialization parameter</td>
</tr>
</tbody>
</table>
This example lists the names of the control files.

```sql
SQL> SELECT NAME FROM V$CONTROLFILE;

NAME
-------------------------------------
/u01/oracle/prod/control01.ctl
/u02/oracle/prod/control02.ctl
/u03/oracle/prod/control03.ctl
```
6. Redo Log File Management

6.1. What Is the Redo Log?

The most crucial structure for recovery operations is the redo log, which consists of two or more pre-allocated files that store all changes made to the database as they occur. Every instance of an Oracle Database has an associated redo log to protect the database in case of an instance failure.

6.2. Redo Threads

When speaking in the context of multiple database instances, the redo log for each database instance is also referred to as a redo thread. In typical configurations, only one database instance accesses an Oracle Database, so only one thread is present. In an Oracle Real Application Clusters environment, however, two or more instances concurrently access a single database and each instance has its own thread of redo. A separate redo thread for each instance avoids contention for a single set of redo log files, thereby eliminating a potential performance bottleneck. The thread number can be assumed to be 1 in all discussions and examples of statements.

6.3. Redo Log Contents

Redo log files are filled with redo records. A redo record, also called a redo entry, is made up of a group of change vectors, each of which is a description of a change made to a single block in the database. For example, if you change a salary value in an employee table, you generate a redo record containing change vectors that describe changes to the data segment block for the table, the undo segment data block, and the transaction table of the undo segments.

Redo entries record data that you can use to reconstruct all changes made to the database, including the undo segments. Therefore, the redo log also protects rollback data. When you recover the database using redo data, the database reads the change vectors in the redo records and applies the changes to the relevant blocks. Redo records are buffered in a circular fashion in the redo log buffer of the SGA and are written to one of the redo log files by the Log Writer (LGWR) database background process. Whenever a transaction is committed, LGWR writes the transaction redo records from the redo log buffer of the SGA to a redo log file, and assigns a system change number (SCN) to identify the redo records for each committed transaction. Only when all redo records associated with a given transaction are safely on disk in the online logs is the user process notified that the transaction has been committed.

Redo records can also be written to a redo log file before the corresponding transaction is committed. If the redo log buffer fills, or another transaction commits, LGWR flushes the entire redo log entries in the redo log buffer to a redo log file, even though some redo records may not be committed. If necessary, the database can roll back these changes.

6.4. How Oracle Database Writes to the Redo Log

The redo log of a database consists of two or more redo log files. The database requires a minimum of two files to guarantee that one is always available for writing while the other is being archived (if the database is in ARCHIVELOG mode). LGWR writes to redo log files in a circular fashion. When the current redo log file fills, LGWR begins writing to the next available redo log file. When the last available redo log file is filled, LGWR returns to the first redo log file and writes to it, starting the cycle again. The numbers next to each line indicate the sequence in which LGWR writes to each redo log file. Filled redo log files are available to LGWR for reuse depending on whether archiving is enabled. If archiving is disabled (the database is in NOARCHIVELOG mode), a filled redo log file is available after the changes recorded in it have been written to the datafiles. If archiving is enabled (the database is in ARCHIVELOG mode), a filled redo log file is available to LGWR after the changes recorded in it have been written to the datafiles and the file has been archived.
6.5. Active (Current) and Inactive Redo Log Files

Oracle Database uses only one redo log file at a time to store redo records written from the redo log buffer. The redo log file that LGWR is actively writing to is called the current redo log file. Redo log files that are required for instance recovery is called active redo log files. Redo log files that are no longer required for instance recovery are called inactive redo log files. If you have enabled archiving (the database is in ARCHIVELOG mode), then the database cannot reuse or overwrite an active online log file until one of the archiver background processes (ARCn) has archived its contents. If archiving is disabled (the database is in NOARCHIVELOG mode), then when the last redo log file is full, LGWR continues by overwriting the first available active file.

6.6. Log Switches and Log Sequence Numbers

A log switch is the point at which the database stops writing to one redo log file and begins writing to another. Normally, a log switch occurs when the current redo log file is completely filled and writing must continue to the next redo log file. However, you can configure log switches to occur at regular intervals, regardless of whether the current redo log file is completely filled. You can also force log switches manually. Oracle Database assigns each redo log file a new log sequence number every time a log switch occurs and LGWR begins writing to it. When the database archives redo log files, the archived log retains its log sequence number. A redo log file that is cycled back for use is given the next available log sequence number. Each online or archived redo log file is uniquely identified by its log sequence number. During crash, instance, or media recovery, the database properly applies redo log files in ascending order by using the log sequence number of the necessary archived and redo log files.

6.7. Planning the Redo Log

This section provides guidelines you should consider when configuring a database instance redo log and contains the following topics:

6.7.1. Multiplexing Redo Log Files

To protect against a failure involving the redo log itself, Oracle Database allows a multiplexed redo log, meaning that two or more identical copies of the redo log can be automatically maintained in separate locations. For the most benefit, these locations should be on separate disks. Even if all copies of the redo log are on the same disk, however, the redundancy can help protect against I/O errors, file corruption, and so on. When redo log files are multiplexed, LGWR concurrently writes the same redo log information to multiple identical redo log files, thereby eliminating a single point of redo log failure. Multiplexing is implemented by creating groups of redo log files. A group consists of a redo log file and its multiplexed copies. Each identical copy is said to be a member of the group. Each redo log group is defined by a number, such as group 1, group 2, and so on.

A_LOG1 and B_LOG1 are both members of Group 1, A_LOG2 and B_LOG2 are both members of Group 2, and so forth. Each member in a group must be exactly the same size. Each member of a log file group is concurrently active—that is, concurrently written to by LGWR—as indicated by the identical log sequence numbers assigned by LGWR. Then it writes concurrently to both A_LOG2 and B_LOG2, and so on. LGWR never writes concurrently to members of different groups (for example, to A_LOG1 and B_LOG2).

Note: Oracle recommends that you multiplex your redo log files. The loss of the log file data can be catastrophic if recovery is required. Note that when you multiplex the redo log, the database must increase the amount of I/O that it performs. Depending on your configuration, this may impact overall database performance.

6.7.2. Responding to Redo Log Failure

Whenever LGWR cannot write to a member of a group, the database marks that member as INVALID and writes an error message to the LGWR trace file and to the database alert log to indicate the problem with the inaccessible files. The specific reaction of LGWR when a redo log member is unavailable depends on the reason for the lack of availability, as summarized in the table that follows.
6.8. Condition LGWR Action

LGWR can successfully write to at least one member in a group available members of a group and ignores the unavailable members. LGWR cannot access the next group at a log switch instance shuts down. In this case, you may need to perform media recovery on the database from the loss of a redo log file. If the database checkpoint has moved beyond the lost redo log, media recovery is not necessary, because the database has saved the data recorded in the redo log to the datafiles. You need only drop the inaccessible redo log group. If the database did not archive the bad log, use ALTER DATABASE CLEAR UNARCHIVED LOG to disable archiving before the log can be dropped.

All members of a group suddenly become inaccessible to LGWR while it instance immediately shuts down. In this case, you need only turn the drive back on and let the database perform automatic instance recovery.

6.9. Placing Redo Log Members on Different Disks

When setting up a multiplexed redo log, place members of a group on different physical disks. If a single disk fails, then only one member of a group becomes unavailable to LGWR and other members remain accessible to LGWR, so the instance can continue to function. If you archive the redo log, spread redo log members across disks to eliminate contention between the LGWR and ARCn background processes. For example, if you have two groups of multiplexed redo log members (a duplexed redo log), place each member on a different disk and set your archiving destination to a fifth disk. Doing so will avoid contention between LGWR (writing to the members) and ARCn (reading the members). Datafiles should also be placed on different disks from redo log files to reduce contention in writing data blocks and redo records.

6.10. Setting the Size of Redo Log Members

When setting the size of redo log files, consider whether you will be archiving the redo log. Redo log files should be sized so that a filled group can be archived to a single unit of offline storage media (such as a tape or disk), with the least amount of space on the medium left unused. For example, suppose only one filled redo log group can fit on a tape and 49% of the tape storage capacity remains unused. In this case, it is better to decrease the size of the redo log files slightly, so that two log groups could be archived on each tape. All members of the same multiplexed redo log group must be the same size. Members of different groups can have different sizes. However, there is no advantage in varying file size between groups. If checkpoints are not set to occur between log switches, make all groups the same size to guarantee that checkpoints occur at regular intervals. The minimum size permitted for a redo log file is 4 MB.

6.11. Creating Redo Log Groups and Members

Plan the redo log of a database and create all required groups and members of redo log files during database creation. However, there are situations where you might want to create additional groups or members. For example, adding groups to a redo log can correct redo log group availability problems.

To create new redo log groups and members, you must have the ALTER DATABASE system privilege. A database can have up to MAXLOGFILES groups.

6.11.1. Creating Redo Log Groups

To create a new group of redo log files, use the SQL statement ALTER DATABASE with the ADD LOGFILE clause. The following statement adds a new group of redo logs to the database:

```
ALTER DATABASE ADD LOGFILE ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo') SIZE 500K;
```
**Note:** Use fully specify filenames of new log members to indicate where the operating system file should be created. Otherwise, the files will be created in either the default or current directory of the database server, depending upon your operating system.

You can also specify the number that identifies the group using the GROUP clause:

```sql
ALTER DATABASE ADD LOGFILE GROUP 10 ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo') SIZE 500K;
```

Using group numbers can make administering redo log groups easier. However, the group number must be between 1 and MAXLOGFILES. Do not skip redo log file group numbers (that is, do not number your groups 10, 20, 30, and so on), or you will consume unnecessary space in the control files of the database.

### 6.11.2. Creating Redo Log Members

In some cases, it might not be necessary to create a complete group of redo log files. A group could already exist, but not be complete because one or more members of the group were dropped (for example, because of a disk failure). In this case, you can add new members to an existing group. To create new redo log members for an existing group, use the SQL statement ALTER DATABASE with the ADD LOGFILE MEMBER clause. The following statement adds a new redo log member to redo log group number 2:

```sql
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2b.rdo' TO GROUP 2;
```

Notice that filenames must be specified, but sizes need not be. The size of the new members is determined from the size of the existing members of the group. When using the ALTER DATABASE statement, you can alternatively identify the target group by specifying all of the other members of the group in the TO clause, as shown in the following example:

```sql
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2c.rdo' TO ('/oracle/dbs/log2a.rdo', '/oracle/dbs/log2b.rdo');
```

Note: Fully specify the filenames of new log members to indicate where the operating system file should be created. Otherwise, the files will be created in either the default or current directory of the database server, depending upon your operating system. You may also note that the status of the new log member is shown as INVALID. This is normal and it will change to active (blank) when it is first used.

### 6.12. Relocating and Renaming Redo Log Members

You can use operating system commands to relocate redo logs, and then use the ALTER DATABASE statement to make their new names (locations) known to the database. This procedure is necessary, for example, if the disk currently used for some redo log files is going to be removed, or if datafiles and a number of redo log files are stored on the same disk and should be separated to reduce contention. To rename redo log members, you must have the ALTER DATABASE system privilege. Additionally, you might also need operating system privileges to copy files to the desired location and privileges to open and back up the database. Before relocating your redo logs, or making any other structural changes to the database, completely back up the database in case you experience problems while performing the operation. As a precaution, after renaming or relocating a set of redo log files, immediately back up the database control file. Use the following steps for relocating redo logs. The example used to illustrate these steps assumes:

- The log files are located on two disks: disk a and disk b.
- The redo log is duplexed: one group consists of the members /diska/logs/log1a.rdo and /diskb/logs/log1b.rdo, and the second group consists of the members /diska/logs/log2a.rdo and /diskb/logs/log2b.rdo.
- The redo log files located on disk a must be relocated to disk c. The new filenames will reflect the new location: /diskc/logs/log1c.rdo and /diskc/logs/log2c.rdo.

#### Steps for Renaming Redo Log Members

1. Using operating system commands, relocate the files to disk c.
2. Use the ALTER DATABASE statement to make the new filenames known to the database.
3. Optionally, back up the database control file.
1. Shut down the database.
   \texttt{SHUTDOWN}

2. Copy the redo log files to the new location.

Operating system files, such as redo log members, must be copied using the appropriate operating system commands. See your operating system specific documentation for more information about copying files.

\textbf{Note:} You can execute an operating system command to copy a file (or perform other operating system commands) without exiting SQL*Plus by using the \texttt{HOST} command. Some operating systems allow you to use a character in place of the word \texttt{HOST}. For example, you can use an exclamation point (!) in UNIX.

The following example uses operating system commands (UNIX) to move the redo log members to a new location:

\begin{verbatim}
mv /diska/logs/log1a.rdo /diskc/logs/log1c.rdo
mv /diska/logs/log2a.rdo /diskc/logs/log2c.rdo
\end{verbatim}

3. Startup the database, mount, but do not open it.

   \texttt{CONNECT / as SYSDBA}

   \texttt{STARTUP MOUNT}

4. Rename the redo log members.

Use the \texttt{ALTER DATABASE} statement with the \texttt{RENAME FILE} clause to rename the database redo log files.

\begin{verbatim}
ALTER DATABASE RENAME FILE '/diska/logs/log1a.rdo', '/diska/logs/log2a.rdo'
   TO '/diskc/logs/log1c.rdo', '/diskc/logs/log2c.rdo';
\end{verbatim}

5. Open the database for normal operation.

The redo log alterations take effect when the database is opened.

\texttt{ALTER DATABASE OPEN;}

\section*{6.13. Dropping Redo Log Groups and Members}

In some cases, you may want to drop an entire group of redo log members. For example, you want to reduce the number of groups in an instance redo log. In a different case, you may want to drop one or more specific redo log members. For example, if a disk failure occurs, you may need to drop all the redo log files on the failed disk so that the database does not try to write to the inaccessible files. In other situations, particular redo log files become unnecessary. For example, a file might be stored in an inappropriate location.

\subsection*{6.13.1. Dropping Log Groups}

To drop a redo log group, you must have the \texttt{ALTER DATABASE} system privilege. Before dropping a redo log group, consider the following restrictions and precautions:

- An instance requires at least two groups of redo log files, regardless of the number of members in the groups. (A group comprises one or more members.)
- You can drop a redo log group only if it is inactive. If you need to drop the current group, first force a log switch to occur. Make sure a redo log group is archived (if archiving is enabled) before dropping it. To see whether this has happened, use the \texttt{V$LOG} view.

\begin{verbatim}
SELECT GROUP#, ARCHIVED, STATUS FROM V$LOG;
\end{verbatim}

\begin{tabular}{ccc}
\hline
GROUP# & ARC & STATUS \\
\hline
 1 & YES & ACTIVE \\
 2 & NO  & CURRENT \\
\hline
\end{tabular}
Drop a redo log group with the SQL statement ALTER DATABASE with the DROP LOGFILE clause.
The following statement drops redo log group number 3:

```sql
ALTER DATABASE DROP LOGFILE GROUP 3;
```

When a redo log group is dropped from the database, and you are not using the Oracle-managed files feature, the operating system files are not deleted from disk. Rather, the control files of the associated database are updated to drop the members of the group from the database structure. After dropping a redo log group, make sure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped redo log files. When using Oracle-managed files, the cleanup of operating systems files is done automatically for you.

### 6.13.2. Dropping Redo Log Members

To drop a redo log member, you must have the ALTER DATABASE system privilege. Consider the following restrictions and precautions before dropping individual redo log members:

It is permissible to drop redo log files so that a multiplexed redo log becomes temporarily asymmetric. For example, if you use duplexed groups of redo log files, you can drop one member of one group, even though all other groups have two members each. However, you should rectify this situation immediately so that all groups have at least two members, and thereby eliminate the single point of failure possible for the redo log.

An instance always requires at least two valid groups of redo log files, regardless of the number of members in the groups. (A group comprises one or more members.) If the member you want to drop is the last valid member of the group, you cannot drop the member until the other members become valid. To see a redo log file status, use the V$LOGFILE view. A redo log file becomes INVALID if the database cannot access it. It becomes STAILE if the database suspects that it is not complete or correct. A stale log file becomes valid again the next time its group is made the active group. You can drop a redo log member only if it is not part of an active or current group. If you want to drop a member of an active group, first force a log switch to occur.

Make sure the group to which a redo log member belongs is archived (if archiving is enabled) before dropping the member. To see whether this has happened, use the V$LOG view. To drop specific inactive redo log members, use the ALTER DATABASE statement with the DROP LOGFILE MEMBER clause.

The following statement drops the redo log `/oracle/dbs/log3c.rdo`:

```sql
ALTER DATABASE DROP LOGFILE MEMBER '/oracle/dbs/log3c.rdo';
```

When a redo log member is dropped from the database, the operating system file is not deleted from disk. Rather, the control files of the associated database are updated to drop the member from the database structure. After dropping a redo log file, make sure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped redo log file.

To drop a member of an active group, you must first force a log switch.

### 6.14. Forcing Log Switches

A log switch occurs when LGWR stops writing to one redo log group and starts writing to another. By default, a log switch occurs automatically when the current redo log file group fills. You can force a log switch to make the currently active group inactive and available for redo log maintenance operations. For example, you want to drop the currently active group, but are not able to do so until the group is inactive. You may also wish to force a log switch if the currently active group needs to be archived at a specific time before the members of the group are completely filled. This option is useful in configurations with large redo log files that take a long time to fill. To force a log switch, you must have the ALTER SYSTEM privilege. Use the ALTER SYSTEM statement with the SWITCH LOGFILE clause.

The following statement forces a log switch:
ALTER SYSTEM SWITCH LOGFILE;

6.15. Clearing a Redo Log File

A redo log file might become corrupted while the database is open, and ultimately stop database activity because archiving cannot continue. In this situation the ALTER DATABASE CLEAR LOGFILE statement can be used to reinitialize the file without shutting down the database. The following statement clears the log files in redo log group number 3:

ALTER DATABASE CLEAR LOGFILE GROUP 3;

This statement overcomes two situations where dropping redo logs are not possible:

- If there are only two log groups
- The corrupt redo log file belongs to the current group
- If the corrupt redo log file has not been archived, use the UNARCHIVED keyword in the statement.

ALTER DATABASE CLEAR UNARCHIVED LOGFILE GROUP 3;

This statement clears the corrupted redo logs and avoids archiving them. The cleared redo logs are available for use even though they were not archived. If you clear a log file that is needed for recovery of a backup, then you can no longer recover from that backup. The database writes a message in the alert log describing the backups from which you cannot recover.

Note: If you clear an unarchived redo log file, you should make another backup of the database.

If you want to clear an unarchived redo log that is needed to bring an offline tablespace online, use the UNRECOVERABLE DATAFILE clause in the ALTER DATABASE CLEAR LOGFILE statement. If you clear a redo log needed to bring an offline tablespace online, you will not be able to bring the tablespace online again. You will have to drop the tablespace or perform an incomplete recovery. Note that tablespaces taken offline normal do not require recovery.

6.16. Redo Log Data Dictionary Views

The following views provide information on redo logs.

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V$LOG</td>
<td>Displays the redo log file information from the control file</td>
</tr>
<tr>
<td>V$LOGFILE</td>
<td>Identifies redo log groups and members and member status</td>
</tr>
<tr>
<td>V$LOG_HISTORY</td>
<td>Contains log history information</td>
</tr>
</tbody>
</table>

The following query returns the control file information about the redo log for a database.

```
SELECT * FROM V$LOG;
GROUP# THREAD# SEQ BYTES MEMBERS ARC STATUS FIRST_CHANGE# FIRST_TIM
-------- ------- ----- ------- -------- -------------- ----------------------
  1       1 10605 1048576 1 YES ACTIVE 11515628 16-APR-00
  2       1 10606 1048576 1 NO CURRENT 11517595 16-APR-00
  3       1 10603 1048576 1 YES INACTIVE 11511666 16-APR-00
  4       1 10604 1048576 1 YES INACTIVE 11513647 16-APR-00
```

To see the names of the entire member of a group, use a query similar to the following:

```
SELECT * FROM V$LOGFILE;
GROUP# STATUS MEMBER
-------- --------- --------------
```
If STATUS is blank for a member, then the file is in use.

1 D:\ORANT\ORADATA\IDDB2\REDO04_LOG
2 D:\ORANT\ORADATA\IDDB2\REDO03_LOG
3 D:\ORANT\ORADATA\IDDB2\REDO02_LOG
4 D:\ORANT\ORADATA\IDDB2\REDO01_LOG
7. Archived Redo Logs Management

7.1. What Is the Archived Redo Log?

Oracle Database lets you save filled groups of redo log files to one or more offline destinations, known collectively as the archived redo log. The process of turning redo log files into archived redo log files is called archiving. This process is only possible if the database is running in ARCHIVELOG mode. You can choose automatic or manual archiving.

An archived redo log file is a copy of one of the filled members of a redo log group. It includes the redo entries and the unique log sequence number of the identical member of the redo log group. For example, if you are multiplexing your redo log, and if group 1 contains identical member files a_log1 and b_log1, then the archiver process (ARCn) will archive one of these member files. Should a_log1 become corrupted, then ARCn can still archive the identical b_log1. The archived redo log contains a copy of every group created since you enabled archiving. When the database is running in ARCHIVELOG mode, the log writer process (LGWR) cannot reuse and hence overwrite a redo log group until it has been archived. The background process ARCn automates archiving operations when automatic archiving is enabled. The database starts multiple archiver processes as needed to ensure that the archiving of filled redo logs does not fall behind. You can use archived redo logs to:

- Recover a database
- Update a standby database
- Get information about the history of a database using the LogMiner utility
- Oracle Data Guard Concepts and Administration discusses setting up and maintaining a standby database
- Oracle Database Utilities contains instructions for using the LogMiner PL/SQL package

7.2. Choosing Between NOARCHIVELOG and ARCHIVELOG Mode

This section describes the issues you must consider when choosing to run your database in NOARCHIVELOG or ARCHIVELOG mode, and contains these topics:

- Running a Database in NOARCHIVELOG Mode
- Running a Database in ARCHIVELOG Mode

The choice of whether to enable the archiving of filled groups of redo log files depends on the availability and reliability requirements of the application running on the database. If you cannot afford to lose any data in your database in the event of a disk failure, use ARCHIVELOG mode. The archiving of filled redo log files can require you to perform extra administrative operations.

7.2.1. Running a Database in NOARCHIVELOG Mode

When you run your database in NOARCHIVELOG mode, you disable the archiving of the redo log. The database control file indicates that filled groups are not required to be archived. Therefore, when a filled group becomes inactive after a log switch, the group is available for reuse by LGWR. NOARCHIVELOG mode protects a database from instance failure but not from media failure. Only the most recent changes made to the database, which are stored in the online redo log groups, are available for instance recovery. If a media failure occurs while the database is in NOARCHIVELOG mode, you can only restore the database to the point of the most recent full database backup. You cannot recover transactions subsequent to that backup.

In NOARCHIVELOG mode you cannot perform online tablespace backups, nor can you use online tablespace backups taken earlier while the database was in ARCHIVELOG mode. To restore a database operating in NOARCHIVELOG mode, you can use only whole database backups taken while the database is closed. Therefore, if you decide to operate a database in NOARCHIVELOG mode, take whole database backups at regular, frequent intervals.
7.2.2. Running a Database in ARCHIVELOG Mode

When you run a database in ARCHIVELOG mode, you enable the archiving of the redo log. The database control file indicates that a group of filled redo log files cannot be reused by LGWR until the group is archived. A filled group becomes available for archiving immediately after a redo log switch occurs. The archiving of filled groups has these advantages:

- A database backup, together with online and archived redo log files, guarantees that you can recover all committed transactions in the event of an operating system or disk failure.
- If you keep an archived log, you can use a backup taken while the database is open and in normal system use.

You can keep a standby database current with its original database by continuously applying the original archived redo logs to the standby. You can configure an instance to archive filled redo log files automatically, or you can archive manually. For convenience and efficiency, automatic archiving is usually best. Figure illustrates how the archiver process (ARC0 in this illustration) writes filled redo log files to the database archived redo log. If all databases in a distributed database operate in ARCHIVELOG mode, you can perform coordinated distributed database recovery. However, if any database in a distributed database is in NOARCHIVELOG mode, recovery of a global distributed database (to make all databases consistent) is limited by the last full backup of any database operating in NOARCHIVELOG mode.

7.3. Controlling Archiving

This section describes how to set the archiving mode of the database and how to control the archiving process.

7.3.1 Setting the Initial Database Archiving Mode

You set the initial archiving mode as part of database creation in the CREATE DATABASE statement. Usually, you can use the default of NOARCHIVELOG mode at database creation because there is no need to archive the redo information generated by that process. After creating the database, decide whether to change the initial archiving mode. If you specify ARCHIVELOG mode, you must have initialization parameters set that specify the destinations for
7.3.2. Changing the Database Archiving Mode

To change the archiving mode of the database, use the ALTER DATABASE statement with the ARCHIVELOG or NOARCHIVELOG clause. To change the archiving mode, you must be connected to the database with administrator privileges (AS SYSDBA). The following steps switch the database archiving mode from NOARCHIVELOG to ARCHIVELOG:

1. Shut down the database instance.

```
SHUTDOWN
```

An open database must first be closed and any associated instances shut down before you can switch the database archiving mode. You cannot change the mode from ARCHIVELOG to NOARCHIVELOG if any datafiles need media recovery.

2. Back up the database.

```
Before making any major change to a database, always back up the database to protect against any problems. This will be your final backup of the database in NOARCHIVELOG mode and can be used if something goes wrong during the change to ARCHIVELOG mode.
```

3. Edit the initialization parameter file to include the initialization parameters that specify the destinations for the archived redo log files.

4. Start a new instance and mount, but do not open the database.

```
STARTUP MOUNT
```

To enable or disable archiving, the database must be mounted but not open.

5. Change the database archiving mode. Then open the database for normal operations.

```
ALTER DATABASE ARCHIVELOG;
ALTER DATABASE OPEN;
```

6. Shut down the database.

```
SHUTDOWN IMMEDIATE
```

7. Back up the database.

Changing the database archiving mode updates the control file. After changing the database archiving mode, you must back up all of your database files and control file. Any previous backup is no longer usable because it was taken in NOARCHIVELOG mode.

7.4. Adjusting the Number of Archiver Processes

The LOG_ARCHIVE_MAX_PROCESSES initialization parameter specifies the number of ARCn processes that the database initially invokes. The default is two processes. There is usually no need specify this initialization parameter or to change its default value, because the database starts additional archiver processes (ARCn) as needed to ensure that the automatic processing of filled redo log files does not fall behind. However, to avoid any runtime overhead of invoking additional ARCn processes, you can set the LOG_ARCHIVE_MAX_PROCESSES initialization parameter to specify up to ten ARCn processes to be started at instance startup. The LOG_ARCHIVE_MAX_PROCESSES parameter is dynamic, and can be changed using the ALTER SYSTEM statement. The database must be mounted but not open. The following statement increases (or decreases) the number of ARCn processes currently running:

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES=3;
```

7.5. Specifying the Archive Destination

Before you can archive redo logs, you must determine the destination to which you will archive and familiarize
yourself with the various destination states. The dynamic performance (V$) views, provide all needed archive information. You can choose whether to archive redo logs to a single destination or multiplex them. If you want to archive only to a single destination, you specify that destination in the LOG_ARCHIVE_DEST initialization parameter. If you want to multiplex the archived logs, you can choose whether to archive to up to ten locations (using the LOG_ARCHIVE_DEST_n parameters) or to archive only to a primary and secondary destination (using LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST). The following table summarizes the multiplexing alternatives, which are further described in the sections that follow.

Method Initialization Parameter Host Example

1. LOG_ARCHIVE_DEST_n Local or LOG_ARCHIVE_DEST_1 = remote 'LOCATION=/disk1/arc'
   where:
   LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1'
   n is an integer from 1 to 10
2. LOG_ARCHIVE_DEST and Local only LOG_ARCHIVE_DEST = '/disk1/arc'
   LOG_ARCHIVE_DUPLEX_DEST LOG_ARCHIVE_DUPLEX_DEST = '/disk2/arc'

Method 1: Using the LOG_ARCHIVE_DEST_n Parameter

Use the LOG_ARCHIVE_DEST_n parameter (where n is an integer from 1 to 10) to specify from one to ten different destinations for archival. Each numerically suffixed parameter uniquely identifies an individual destination.

Keyword Indicates Example

LOCATION A local file system   LOG_ARCHIVE_DEST_1 = location. 'LOCATION=/disk1/arc'
SERVICE Remote archival   LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1' through Oracle Net service name.

If you use the LOCATION keyword, specify a valid path name for your operating system. If you specify SERVICE, the database translates the net service name through the tnsnames.ora file to a connect descriptor. The descriptor contains the information necessary for connecting to the remote database. The service name must have an associated database SID, so that the database correctly updates the log history of the control file for the standby database. Perform the following steps to set the destination for archived redo logs using the LOG_ARCHIVE_DEST_n initialization parameter:
1. Use SQL*Plus to shut down the database.
   SHUTDOWN
2. Set the LOG_ARCHIVE_DEST_n initialization parameter to specify from one to ten archiving locations. The LOCATION keyword specifies an operating system specific path name. For example, enter:

   LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/archive'
   LOG_ARCHIVE_DEST_2 = 'LOCATION = /disk2/archive'
   LOG_ARCHIVE_DEST_3 = 'LOCATION = /disk3/archive'

   If you are archiving to a standby database, use the SERVICE keyword to specify a valid net service name from the tnsnames.ora file. For example, enter:

   LOG_ARCHIVE_DEST_4 = 'SERVICE = standby1'

3. Optionally, set the LOG_ARCHIVE_FORMAT initialization parameter, using %t to include the thread number as part of the file name, %s to include the log sequence number, and %r to include the resetlogs ID (a timestamp value represented in ub4). Use capital letters (%T, %S, and %R) to pad the file name to the left with zeroes.

   Note: If the COMPATIBLE initialization parameter is set to 10.0.0 or higher, the database requires the specification of resetlogs ID (%) when you include the LOG_ARCHIVE_FORMAT parameter. The default for this parameter is operating system dependent. For example, this is the default format for UNIX:

   LOG_ARCHIVE_FORMAT=%t_%s_%r.dbf
The incarnation of a database changes when you open it with the RESETLOGS option. Specifying %r causes the database to capture the resetlogs ID in the archived redo log file name.

The following example shows a setting of LOG_ARCHIVE_FORMAT:

LOG_ARCHIVE_FORMAT = arch_%t_%s_%r.arc

This setting will generate archived logs as follows for thread 1; log sequence numbers 100, 101, and 102; resetlogs ID 509210197. The identical resetlogs ID indicates that the files are all from the same database incarnation:

/disk1/archive/arch_1_100_509210197.arc,
/disk1/archive/arch_1_101_509210197.arc,
/disk1/archive/arch_1_102_509210197.arc
/disk2/archive/arch_1_100_509210197.arc,
/disk2/archive/arch_1_101_509210197.arc,
/disk2/archive/arch_1_102_509210197.arc
/disk3/archive/arch_1_100_509210197.arc,
/disk3/archive/arch_1_101_509210197.arc,
/disk3/archive/arch_1_102_509210197.arc

Method 2: Using LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST

To specify a maximum of two locations, use the LOG_ARCHIVE_DEST parameter to specify a primary archive destination and the LOG_ARCHIVE_DUPLEX_DEST to specify an optional secondary archive destination. All locations must be local. Whenever the database archives a redo log, it archives it to every destination specified by either set of parameters.

Perform the following steps to use method 2:
1. Use SQL*Plus to shut down the database.

SHUTDOWN

2. Specify destinations for the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameter (you can also specify LOG_ARCHIVE_DUPLEX_DEST dynamically using the ALTER SYSTEM statement).

For example, enter:

LOG_ARCHIVE_DEST = '/disk1/archive'
LOG_ARCHIVE_DUPLEX_DEST = '/disk2/archive'

3. Set the LOG_ARCHIVE_FORMAT initialization parameter as described in step 3 for method 1.

7.6. Understanding Archive Destination Status

Each archive destination has the following variable characteristics that determine its status:

- Valid / Invalid: indicates whether the disk location or service name information is specified and valid
- Enabled/Disabled: indicates the availability state of the location and whether the database can use the destination
- Active/Inactive: indicates whether there was a problem accessing the destination
- Several combinations of these characteristics are possible. To obtain the current status and other information about each destination for an instance, query the V$ARCHIVE_DEST view.
- The LOG_ARCHIVE_DEST_STATE_n (where n is an integer from 1 to 10) initialization parameter lets you control the availability state of the specified destination (n).
- ENABLE indicates that the database can use the destination.
- DEFER indicates that the location is temporarily disabled.
• ALTERNATE indicates that the destination is an alternate.
• The availability state of the destination is DEFER, unless there is a failure of its parent destination, in which case its state becomes ENABLE.

7.7. Managing Archive Destination Failure

Sometimes archive destinations can fail, causing problems when you operate in automatic archiving mode. Oracle Database provides procedures to help you minimize the problems associated with destination failure. These procedures are discussed in the sections that follow:

• Specifying the Minimum Number of Successful Destinations
• Rearchiving to a Failed Destination

7.7.1. Specifying the Minimum Number of Successful Destinations

The optional initialization parameter LOG_ARCHIVE_MIN_SUCCEED_DEST=n determines the minimum number of destinations to which the database must successfully archive a redo log group before it can reuse online log files. The default value is 1. Valid values for n are 1 to 2 if you are using duplexing or 1 to 10 if you are multiplexing.

7.7.2. Specifying Mandatory and Optional Destinations

The LOG_ARCHIVE_DEST_n parameter lets you specify whether a destination is OPTIONAL (the default) or MANDATORY. The LOG_ARCHIVE_MIN_SUCCEED_DEST=n parameter uses all MANDATORY destinations plus some number of non-standby OPTIONAL destinations to determine whether LGWR can overwrite the online log. The following rules apply:

• Omitting the MANDATORY attribute for a destination is the same as specifying OPTIONAL.
• You must have at least one local destination, which you can declare OPTIONAL or MANDATORY.
• When you specify a value for LOG_ARCHIVE_MIN_SUCCEED_DEST=n, Oracle Database will treat at least one local destination as MANDATORY, because the minimum value for LOG_ARCHIVE_MIN_SUCCEED_DEST is 1.
• If any MANDATORY destination fails, including a MANDATORY standby destination, Oracle Database ignores the LOG_ARCHIVE_MIN_SUCCEED_DEST parameter.
• The LOG_ARCHIVE_MIN_SUCCEED_DEST value cannot be greater than the number of destinations, nor can it be greater than the number of MANDATORY destinations plus the number of OPTIONAL local destinations.
• If you DEFER a MANDATORY destination, and the database overwrites the online log without transferring the archived log to the standby site, then you must transfer the log to the standby manually.

If you are duplexing the archived logs, you can establish which destinations are mandatory or optional by using the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameters. The following rules apply:

• Any destination declared by LOG_ARCHIVE_DEST is mandatory.
• Any destination declared by LOG_ARCHIVE_DUPLEX_DEST is optional if LOG_ARCHIVE_MIN_SUCCEED_DEST = 1 and mandatory if LOG_ARCHIVE_MIN_SUCCEED_DEST = 2.

Specifying the Number of Successful Destinations: Scenarios

You can see the relationship between the LOG_ARCHIVE_DEST_n and LOG_ARCHIVE_MIN_SUCCEEDDEST parameters most easily through sample scenarios.
• The database can reuse log files only if at least one of the OPTIONAL destinations succeeds.
• The database can reuse log files only if at least two of the OPTIONAL destinations succeed.
• The database can reuse log files only if all of the OPTIONAL destinations succeed.

This scenario shows that even though you do not explicitly set any of your destinations to MANDATORY using the LOG_ARCHIVE_DEST_n parameter, the database must successfully archive to one or more of these locations when LOG_ARCHIVE_MIN_SUCCEED_DEST is set to 1, 2, or 3.
8. Tablespaces Management

8.1. Using Multiple Tablespaces

Using multiple tablespaces allows you more flexibility in performing database operations. When a database has multiple tablespaces, you can:

- Separate user data from data dictionary data to reduce I/O contention.
- Separate data of one application from the data of another to prevent multiple applications from being affected if a tablespace must be taken offline.
- Store the datafiles of different tablespaces on different disk drives to reduce I/O contention. Take individual tablespaces offline while others remain online, providing better overall availability. Optimizing tablespace use by reserving a tablespace for a particular type of database use, such as high update activity, read-only activity, or temporary segment storage.
- Back up individual tablespaces. Some operating systems set a limit on the number of files that can be open simultaneously. Such limits can affect the number of tablespaces that can be simultaneously online.

To avoid exceeding your operating system limit, plan your tablespaces efficiently. Create only enough tablespaces to fulfill your needs, and create these tablespaces with as few files as possible. If you need to increase the size of a tablespace, add one or two large datafiles, or create datafiles with autoextension enabled, rather than creating many small datafiles. Review your data in light of these factors and decide how many tablespaces you need for your database design.

8.2. Creating Tablespaces

Before you can create a tablespace, you must create a database to contain it. The primary tablespace in any database is the SYSTEM tablespace, which contains information basic to the functioning of the database server, such as the data dictionary and the system rollback segment. The SYSTEM tablespace is the first tablespace created at database creation. It is managed as any other tablespace, but requires a higher level of privilege and is restricted in some ways. For example, you cannot rename or drop the SYSTEM tablespace or take it offline. The SYSAUX tablespace, which acts as an auxiliary tablespace to the SYSTEM tablespace, is also always created when you create a database. It contains information about and the schemas used by various Oracle products and features, so that those products do not require their own tablespaces. As for the SYSTEM tablespace, management of the SYSAUX tablespace requires a higher level of security and you cannot rename or drop it.

The steps for creating tablespaces vary by operating system, but the first step is always to use your operating system to create a directory structure in which your datafiles will be allocated. On most operating systems, you specify the size and fully specified filenames of datafiles when you create a new tablespace or alter an existing tablespace by adding datafiles. Whether you are creating a new tablespace or modifying an existing one, the database automatically allocates and formats the datafiles as specified. To create a new tablespace, use the SQL statement CREATE TABLESPACE or CREATE TEMPORARY TABLESPACE. You must have the CREATE TABLESPACE system privilege to create a tablespace. Later, you can use the ALTER TABLESPACE or ALTER DATABASE statements to alter the tablespace. You must have the ALTER TABLESPACE or ALTER DATABASE system privilege, correspondingly. You can also use the CREATE UNDO TABLESPACE statement to create a special type of tablespace called an undo tablespace, which is specifically designed to contain undo records. These are records generated by the database that are used to roll back, or undo, changes to the database for recovery, read consistency, or as requested by a ROLLBACK statement. The creation and maintenance of permanent and temporary tablespaces are discussed in the following sections:

- Locally Managed Tablespaces
- Bigfile Tablespaces
Locally managed tablespaces track all extent information in the tablespace itself by using bitmaps, resulting in the following benefits: Fast, concurrent space operations. Space allocations and deallocations modify locally managed resources (bitmaps stored in header files).

Readable standby databases are allowed, because locally managed temporary tablespaces do not generate any undo or redo. Space allocation is simplified, because when the AUTOALLOCATE clause is specified, the database automatically selects the appropriate extent size. User reliance on the data dictionary is reduced, because the necessary information is stored in file headers and bitmap blocks. Coalescing free extents is unnecessary for locally managed tablespaces. All tablespaces, including the SYSTEM tablespace, can be locally managed. The DBMS_SPACE_ADMIN package provides maintenance procedures for locally managed tablespaces.

8.2.1. Creating a Locally Managed Tablespace

Create a locally managed tablespace by specifying LOCAL in the EXTENT MANAGEMENT clause of the CREATE TABLESPACE statement. This is the default for new permanent tablespaces, but you must specify the EXTENT MANAGEMENT LOCAL clause if you want to specify either the AUTOALLOCATE clause or the UNIFORM clause. You can have the database manage extents for you automatically with the AUTOALLOCATE clause (the default), or you can specify that the tablespace is managed with uniform extents of a specific size (UNIFORM).

If you expect the tablespace to contain objects of varying sizes requiring many extents with different extent sizes, then AUTOALLOCATE is the best choice. AUTOALLOCATE is also a good choice if it is not important for you to have a lot of control over space allocation and deallocation, because it simplifies tablespace management. Some space may be wasted with this setting, but the benefit of having Oracle Database manage your space most likely outweighs this drawback. If you want exact control over unused space and you can predict exactly the space to be allocated for an object or objects and the number and size of extents, then UNIFORM is a good choice. This setting ensures that you will never have unusable space in your tablespace. When you do not explicitly specify the type of extent management, Oracle Database determines extent management as follows:

- If the CREATE TABLESPACE statement omits the DEFAULT storage clause, then the database creates a locally managed autoallocated tablespace.
- If the CREATE TABLESPACE statement includes a DEFAULT storage clause, then the database considers the following:
  - If you specified the MINIMUM EXTENT clause, the database evaluates whether the values of MINIMUM EXTENT, INITIAL, and NEXT are equal and the value of PCTINCREASE is 0. If so, the database creates a locally managed uniform tablespace with extent size = INITIAL. If the MINIMUM EXTENT, INITIAL, and NEXT parameters are not equal, or if PCTINCREASE is not 0, the database ignores any extent storage parameters you may specify and creates a locally managed, autoallocated tablespace.
  - If you did not specify MINIMUM EXTENT clause, the database evaluates only whether the storage values of INITIAL and NEXT are equal and PCTINCREASE is 0. If so, the tablespace is locally managed and uniform. Otherwise, the tablespace is locally managed and autoallocated. The following statement creates a locally managed tablespace named lmtbsb and specifies AUTOALLOCATE:

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

AUTOALLOCATE causes the tablespace to be system managed with a minimum extent size of 64K.
The alternative to AUTOALLOCATE is UNIFORM. Which specifies that the tablespace is managed with extents of uniform size. You can specify that size in the SIZE clause of UNIFORM. If you omit SIZE, then the default size is 1M. The following example creates a tablespace with uniform 128K extents. (In a database with 2K blocks, each extent would be equivalent to 64 database blocks). Each 128K extent is represented by a bit in the extent bitmap for this file.

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M EXTERNAL MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

You cannot specify the DEFAULT storage clause, MINIMUM EXTENT, or TEMPORARY when you explicitly specify EXTERNAL MANAGEMENT LOCAL. If you want to create a temporary locally managed tablespace, use the CREATE TEMPORARY TABLESPACE statement.

Note: When you allocate a datafile for a locally managed tablespace, you should allow space for metadata used for space management (the extent bitmap or space header segment) which are part of user space. For example, if you specify the UNIFORM clause in the extent management clause but you omit the SIZE parameter, then the default extent size is 1MB. In that case, the size specified for the datafile must be larger (at least one block plus space for the bitmap) than 1MB.

### 8.2.2. Specifying Segment Space Management in Locally Managed Tablespaces

In a locally managed tablespace, there are two methods that Oracle Database can use to manage segment space: automatic and manual. Manual segment space management uses linked lists called "freelists" to manage free space in the segment, while automatic segment space management uses bitmaps. Automatic segment space management is the more efficient method, and is the default for all new permanent, locally managed tablespaces.

Automatic segment space management delivers better space utilization than manual segment space management. It is also self-tuning, in that it scales with increasing number of users or instances. In an Oracle Real Application Clusters environment, automatic segment space management allows for a dynamic affinity of space to instances. In addition, for many standard workloads, application performance with automatic segment space management is better than the performance of a well-tuned application using manual segment space management. Although automatic segment space management is the default for all new permanent, locally managed tablespaces, you can explicitly enable it with the SEGMENT SPACE MANAGEMENT AUTO clause.

The following statement creates tablespace lmtbsb with automatic segment space management:

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M EXTERNAL MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;
```

The SEGMENT SPACE MANAGEMENT MANUAL clause disables automatic segment space management. The segment space management that you specify at tablespace creation time applies to all segments subsequently created in the tablespace. You cannot change the segment space management mode of a tablespace.

**Note:**

- If you set extent management to LOCAL UNIFORM, then you must ensure that each extent contains at least 5 database blocks.
- If you set extent management to LOCAL AUTOALLOCATE, and if the database block size is 16K or greater, then Oracle manages segment space by creating extents with a minimum size of 5 blocks rounded up to 64K.
- Locally managed tablespaces using automatic segment space management can be created as single-file or bigfile tablespaces.

### 8.3. Bigfile Tablespaces

A bigfile tablespace is a tablespace with a single, but very large (up to 4G blocks) datafile. Traditional smallfile tablespaces, in contrast, can contain multiple datafiles, but the files cannot be as large. The benefits of bigfile
tablespaces are the following:

- A bigfile tablespace with 8K blocks can contain a 32 terabyte datafile. A bigfile tablespace with 32K blocks can contain a 128 terabyte datafile. The maximum number of datafiles in an Oracle Database is limited (usually to 64K files). Therefore, bigfile tablespaces can significantly enhance the storage capacity of an Oracle Database.

- Bigfile tablespaces can reduce the number of datafiles needed for a database. An additional benefit is that the DB_FILES initialization parameter and MAXDATAFILES parameter of the CREATE DATABASE and CREATE CONTROLFILE statements can be adjusted to reduce the amount of SGA space required for datafile information and the size of the control file.

- Bigfile tablespaces simplify database management by providing datafile transparency. SQL syntax for the ALTER TABLESPACE statement lets you perform operations on tablespaces, rather than the underlying individual datafiles. Bigfile tablespaces are supported only for locally managed tablespaces with automatic segment space management, with three exceptions: locally managed undo tablespaces, temporary tablespaces, and the SYSTEM tablespace.

**Note:**

Bigfile tablespaces are intended to be used with Automatic Storage Management (ASM) or other logical volume managers that supports striping or RAID, and dynamically extensible logical volumes. Avoid creating bigfile tablespaces on a system that does not support striping because of negative implications for parallel query execution and RMAN backup parallelization. Using bigfile tablespaces on platforms that do not support large file sizes is not recommended and can limit tablespace capacity. Refer to your operating system specific documentation for information about maximum supported file sizes.

### 8.3.1. Creating a Bigfile Tablespace

To create a bigfile tablespace, specify the BIGFILE keyword of the CREATE TABLESPACE statement (CREATE BIGFILE TABLESPACE ...). Oracle Database automatically creates a locally managed tablespace with automatic segment space management. You can, but need not, specify EXTENT MANAGEMENT LOCAL and SEGMENT SPACE MANAGEMENT AUTO in this statement. However, the database returns an error if you specify EXTENT MANAGEMENT DICTIONARY or SEGMENT SPACE MANAGEMENT MANUAL. The remaining syntax of the statement is the same as for the CREATE TABLESPACE statement, but you can only specify one datafile. For example:

```
CREATE BIGFILE TABLESPACE bigtbs DATAFILE '/u02/oracle/data/bigtbs01.dbf' SIZE 50G
```

You can specify SIZE in kilobytes (K), megabytes (M), gigabytes (G), or terabytes (T). If the default tablespace type was set to BIGFILE at database creation, you need not specify the keyword BIGFILE in the CREATE TABLESPACE statement. A bigfile tablespace is created by default. If the default tablespace type was set to BIGFILE at database creation, but you want to create a traditional (smallfile) tablespace, then specify a CREATE SMALLFILE TABLESPACE statement to override the default tablespace type for the tablespace that you are creating.

### Identifying a Bigfile Tablespace

The following views contain a BIGFILE column that identifies a tablespace as a bigfile tablespace:

- DBA_TABLESPACES
- USER_TABLESPACES
- V$TABLESPACE

You can also identify a bigfile tablespace by the relative file number of its single datafile. That number is 1024 on most platforms, but 4096 on OS/390.
8.4. Encrypted Tablespaces

You can encrypt any permanent tablespace to protect sensitive data. Tablespace encryption is completely transparent to your applications, so no application modification is necessary. Encrypted tablespaces primarily protect your data from unauthorized access by means other than through the database. For example, when encrypted tablespaces are written to backup media for travel from one Oracle database to another or for travel to an off-site facility for storage, they remain encrypted. Also, encrypted tablespaces protect data from users who try to circumvent the security features of the database and access database files directly through the operating system file system.

Tablespace encryption does not address all security issues. It does not, for example, provide access control from within the database. Any user who is granted privileges on objects stored in an encrypted tablespace can access those objects without providing any kind of additional password or key. When you encrypt a tablespace, all tablespace blocks are encrypted. All segment types are supported for encryption, including tables, clusters, indexes, LOBs (BASICFILE and SECUREFILE), table and index partitions, and so on.

To maximize security, data from an encrypted tablespace is automatically encrypted when written to the undo tablespace, to the redo logs, and to any temporary tablespace. There is no need to explicitly create encrypted undo or temporary tablespaces, and in fact, you cannot specify encryption for those tablespace types. For partitioned tables and indexes that have different partitions in different tablespaces, it is permitted to use both encrypted and non-encrypted tablespaces in the same table or index. Tablespace encryption uses the transparent data encryption feature of Oracle Database, which requires that you create an Oracle wallet to store the master encryption key for the database. The wallet must be open before you can create the encrypted tablespace and before you can store or retrieve encrypted data. When you open the wallet, it is available to all session, and it remains open until you explicitly close it or until the database is shut down.

To encrypt a tablespace, you must open the database with the COMPATIBLE initialization parameter set to 11.1.0 or higher. The default setting for COMPATIBLE for a new Oracle Database 11g Release 1 installation is 11.1.0. Any user who can create a tablespace can create an encrypted tablespace. Transparent data encryption supports industry-standard encryption algorithms, including the following Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) algorithms:

- 3DES168
- AES128
- AES192
- AES256

The encryption key length is implied by the algorithm name. For example, the AES128 algorithm uses 128-bit keys. You specify the algorithm to use when you create the tablespace, and different tablespaces can use different algorithms. Although longer key lengths theoretically provide greater security, there is a trade-off in CPU overhead. If you do not specify the algorithm in your CREATE TABLESPACE statement, AES128 is the default. There is no disk space overhead for encrypting a tablespace.

Examples
The following statement creates an encrypted tablespace with the default encryption algorithm:

```
CREATE TABLESPACE securespace
DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf' SIZE 100M
ENCRYPTION DEFAULT STORAGE(ENCRYPT);
```

The following statement creates the same tablespace with the AES256 algorithm:

```
CREATE TABLESPACE securespace
DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf' SIZE 100M
ENCRYPTION USING 'AES256' DEFAULT STORAGE(ENCRYPT);
```
8.4.1. Restrictions
The following are restrictions for encrypted tablespaces:

- You cannot encrypt an existing tablespace with an ALTER TABLESPACE statement.
- However, you can use Data Pump or SQL statements such as CREATE TABLE AS SELECT or ALTER TABLE MOVE to move existing table data into an encrypted tablespace.
- Encrypted tablespaces are subject to restrictions when transporting to another database.
- When recovering a database with encrypted tablespaces (for example after a SHUTDOWN ABORT or a catastrophic error that brings down the database instance), you must open the Oracle wallet after database mount and before database open, so the recovery process can decrypt data blocks and redo.

8.4.2. Querying Tablespace Encryption Information
The DBA_TABLESPACES and USER_TABLESPACES data dictionary views include a column named ENCRYPTED. This column contains YES for encrypted tablespaces. The view V$ENCRYPTED_TABLESPACES lists all currently encrypted tablespaces. The following query displays the name and encryption algorithm of encrypted tablespaces:

```
SELECT t.name, e.encryptionalgorithm FROM v$tablespace t, $encrypted_tablespaces e WHERE t.ts# = e.ts#;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>ALGORITHM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURESPACE</td>
<td>AES128</td>
</tr>
</tbody>
</table>

8.5. Temporary Tablespaces
A temporary tablespace contains transient data that persists only for the duration of the session. Temporary tablespaces can improve the concurrency of multiple sort operations that do not fit in memory and can improve the efficiency of space management operations during sorts. Temporary tablespaces are used to store the following:

- Intermediate sort results
- Temporary tables and temporary indexes
- Temporary LOBs
- Temporary B-trees

Within a temporary tablespace, all sort operations for a particular instance share a single sort segment, and sort segments exist for every instance that performs sort operations that require temporary space. A sort segment is created by the first statement after startup that uses the temporary tablespace for sorting, and is released only at shutdown. By default, a single temporary tablespace named TEMP is created for each new Oracle Database installation. You can create additional temporary tablespaces with the CREATE TABLESPACE statement. You can assign a temporary tablespace to each database user with the CREATE USER or ALTER USER statement. A single temporary tablespace can be shared by multiple users. You cannot explicitly create objects in a temporary tablespace.

Note: The exception to the preceding statement is a temporary table. When you create a temporary table, its rows are stored in your default temporary tablespace, unless you create the table in a new temporary tablespace.

8.5.1. Default Temporary Tablespace
Users who are not explicitly assigned a temporary tablespace use the database default temporary tablespace, which for new installations is TEMP. You can change the default temporary tablespace for the database with the following command:

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tablespace_name;
```
To determine the current default temporary tablespace for the database, run the following query:

```
SELECT PROPERTY_NAME, PROPERTY_VALUE FROM DATABASE_PROPERTIES WHERE
  PROPERTY_NAME='DEFAULT_TEMP_TABLESPACE';
```

<table>
<thead>
<tr>
<th>PROPERTY_NAME</th>
<th>PROPERTY_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_TEMP_TABLESPACE</td>
<td>TEMP</td>
</tr>
</tbody>
</table>

### 8.5.2. Space Allocation in a Temporary Tablespace

You can view the allocation and deallocation of space in a temporary tablespace sort segment using the V$SORT_SEGMENT view. The V$SORT_USAGE view identifies the current sort users in those segments. When a sort operation that uses temporary space completes, allocated extents in the sort segment are not deallocated; they are just marked as free and available for reuse. The DBA_TEMP_FREE_SPACE view displays the total allocated and free space in each temporary tablespace. You can manually shrink a locally managed temporary tablespace that has a large amount of unused space.

### 8.5.3. Creating a Locally Managed Temporary Tablespace

Because space management is much simpler and more efficient in locally managed tablespaces, they are ideally suited for temporary tablespaces. Locally managed temporary tablespaces use tempfiles, which do not modify data outside of the temporary tablespace or generate any redo for temporary tablespace data. Because of this, they enable you to perform on-disk sorting operations in a read-only or standby database. You also use different views for viewing information about tempfiles than you would for datafiles. The V$TEMPFILE and DBA_TEMP_FILES views are analogous to the V$DATAFILE and DBA_DATA_FILES views. To create a locally managed temporary tablespace, you use the CREATE TEMPORARY TABLESPACE statement, which requires that you have the CREATE TABLESPACE System privilege.

The following statement creates a temporary tablespace in which each extent is 16M. Each 16M extent (which is the equivalent of 8000 blocks when the standard block size is 2K) is represented by a bit in the bitmap for the file.

```
CREATE TEMPORARY TABLESPACE lmtemp TEMPFILE '/u02/oracle/data/lmtemp01.dbf'
  SIZE 20M REUSE EXTENT MANAGEMENT LOCAL UNIFORM SIZE 16M;
```

The extent management clause is optional for temporary tablespaces because all temporary tablespaces are created with locally managed extents of a uniform size. The default for SIZE is 1M. But if you want to specify another value for SIZE, you can do so as shown in the preceding statement.

**Note:** On some operating systems, the database does not allocate space for the tempfile until the tempfile blocks are actually accessed. This delay in space allocation results in faster creation and resizing of tempfiles, but it requires that sufficient disk space is available when the tempfiles are later used. Please refer to your operating system documentation to determine whether the database allocates tempfile space in this way on your system.

### 8.5.4 Creating a Bigfile Temporary Tablespace

Just as for regular tablespaces, you can create single-file (bigfile) temporary tablespaces. Use the CREATE BIGFILE TEMPORARY TABLESPACE statement to create a single-tempfile tablespace.

### Viewing Space Usage for Temporary Tablespaces

The DBA_TEMP_FREE_SPACE dictionary view contains information about space usage for each temporary tablespace. The information includes the space allocated and the free space. You can query this view for these statistics using the following command.

```
SELECT * from DBA_TEMP_FREE_SPACE;
```

<table>
<thead>
<tr>
<th>TABLESPACE_NAME</th>
<th>TABLESPACE_SIZE</th>
<th>ALLOCATED_SPACE</th>
<th>FREE_SPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMP</td>
<td>250609664</td>
<td>250609664</td>
<td></td>
</tr>
</tbody>
</table>
8.5.4. Multiple Temporary Tablespaces: Using Tablespace Groups

A tablespace group enables a user to consume temporary space from multiple tablespaces. Using a tablespace group, rather than a single temporary tablespace, can alleviate problems caused where one tablespace is inadequate to hold the results of a sort, particularly on a table that has many partitions. A tablespace group enables parallel execution servers in a single parallel operation to use multiple temporary tablespaces. A tablespace group has the following characteristics:

- It contains at least one tablespace. There is no explicit limit on the maximum number of tablespaces that are contained in a group.
- It shares the namespace of tablespaces, so its name cannot be the same as any tablespace.
- You can specify a tablespace group name wherever a tablespace name would appear when you assign a default temporary tablespace for the database or a temporary tablespace for a user.

You do not explicitly create a tablespace group. Rather, it is created implicitly when you assign the first temporary tablespace to the group. The group is deleted when the last temporary tablespace it contains is removed from it. The view DBA_TABLESPACE_GROUPS lists tablespace groups and their member tablespaces.

8.6. Creating a Tablespace Group

You create a tablespace group implicitly when you include the TABLESPACE GROUP clause in the CREATE TEMPORARY TABLESPACE or ALTER TABLESPACE statement and the specified tablespace group does not currently exist. For example, if neither group1 nor group2 exists, then the following statements create those groups, each of which has only the specified tablespace as a member:

```
CREATE TEMPORARY TABLESPACE lmtemp2 TEMPFILE '/u02/oracle/data/lmtemp201.dbf'
SIZE 50M TABLESPACE GROUP group1;

ALTER TABLESPACE lmtemp TABLESPACE GROUP group2;
```

8.6.1. Changing Members of a Tablespace Group

You can add a tablespace to an existing tablespace group by specifying the existing group name in the TABLESPACE GROUP clause of the CREATE TEMPORARY TABLESPACE or ALTER TABLESPACE statement. The following statement adds a tablespace to an existing group. It creates and adds tablespace lmtemp3 to group1, so that group1 contains tablespaces lmtemp2 and lmtemp3.

```
CREATE TEMPORARY TABLESPACE lmtemp3 TEMPFILE '/u02/oracle/data/lmtemp301.dbf'
SIZE 25M TABLESPACE GROUP group1;
```

The following statement also adds a tablespace to an existing group, but in this case because tablespace lmtemp2 already belongs to group1, it is in effect moved from group1 to group2:

```
ALTER TABLESPACE lmtemp2 TABLESPACE GROUP group2;
```

Now group2 contains both lmtemp and lmtemp2, while group1 consists of only lmtemp3. You can remove a tablespace from a group as shown in the following statement:

```
ALTER TABLESPACE lmtemp3 TABLESPACE GROUP '';
```

Tablespace lmtemp3 no longer belongs to any group. Further, since there are no longer any members of group1, this results in the implicit deletion of group1.

Assigning a Tablespace Group as the Default Temporary Tablespace
Use the `ALTER DATABASE ...DEFAULT TEMPORARY TABLESPACE` statement to assign a tablespace group as the default temporary tablespace for the database. For example:

```
ALTER DATABASE sample DEFAULT TEMPORARY TABLESPACE group2;
```

Any user who has not explicitly been assigned a temporary tablespace will now use tablespaces `lmtemp` and `lmtemp2`. If a tablespace group is specified as the default temporary tablespace, you cannot drop any of its member tablespaces. You must first remove the tablespace from the tablespace group. Likewise, you cannot drop a single temporary tablespace as long as it is the default temporary tablespace.

### 8.7. Specifying Nonstandard Block Sizes for Tablespaces

You can create tablespaces with block sizes different from the standard database block size, which is specified by the `DB_BLOCK_SIZE` initialization parameter. This feature lets you transport tablespaces with unlike block sizes between databases. Use the `BLOCKSIZE` clause of the `CREATE TABLESPACE` statement to create a tablespace with a block size different from the database standard block size. In order for the `BLOCKSIZE` clause to succeed, you must have already set the `DB_CACHE_SIZE` and at least one `DB_nK_CACHE_SIZE` initialization parameter. Further, the integer you specify in the `BLOCKSIZE` clause must correspond with the setting of one `DB_nK_CACHE_SIZE` parameter setting. Although redundant, specifying a `BLOCKSIZE` equal to the standard block size, as specified by the `DB_BLOCK_SIZE` initialization parameter, is allowed. The following statement creates tablespace `lmtbsb`, but specifies a block size that differs from the standard database block size (as specified by the `DB_BLOCK_SIZE` initialization parameter):

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K BLOCKSIZE 8K;
```

### 8.8. Altering Tablespace Availability

You can take an online tablespace offline so that it is temporarily unavailable for general use. The rest of the database remains open and available for users to access data. Conversely, you can bring an offline tablespace online to make the schema objects within the tablespace available to database users. The database must be open to alter the availability of a tablespace. To alter the availability of a tablescape, use the `ALTER TABLESPACE` statement. You must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

#### 8.8.1. Taking Tablespaces Offline

You may want to take a tablespace offline for any of the following reasons:
- To make a portion of the database unavailable while allowing normal access to the remainder of the database
- To perform an offline tablespace backup (even though a tablespace can be backed up while online and in use)
- To make an application and its group of tables temporarily unavailable while updating or maintaining the application

#### 8.8.2. To rename or relocate tablespace datafiles

When a tablespace is taken offline, the database takes all the associated files offline. You cannot take the following tablespaces offline:

- **SYSTEM**
- The undo tablespace
- Temporary tablespaces

Before taking a tablespace offline, consider altering the tablespace allocation of any users who have been assigned the tablespace as a default tablespace. Doing so is advisable because those users will not be able to access objects in the tablespace while it is offline. You can specify any of the following parameters as part of the `ALTER TABLESPACE...OFFLINE` statement:
### Tablespace Management

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL</td>
<td>A tablespace can be taken offline normally if no error conditions exist for any of the datafiles of the tablespace. No datafile in the tablespace can be currently offline as the result of a write error. When you specify OFFLINE NORMAL, the database takes a checkpoint for all datafiles of the tablespace as it takes them offline. NORMAL is the default.</td>
</tr>
<tr>
<td>TEMPORARY</td>
<td>A tablespace can be taken offline temporarily, even if there are error conditions for one or more files of the tablespace. When you specify OFFLINE TEMPORARY, the database takes offline the datafiles that are not already offline, check pointing them as it does so. If no files are offline, but you use the temporary clause, media recovery is not required to bring the tablespace back online. However, if one or more files of the tablespace are offline because of write errors, and you take the tablespace offline temporarily, the tablespace requires recovery before you can bring it back online.</td>
</tr>
<tr>
<td>IMMEDIATE</td>
<td>A tablespace can be taken offline immediately, without the database taking a checkpoint on any of the datafiles. When you specify OFFLINE IMMEDIATE, media recovery for the tablespace is required before the tablespace can be brought online. You cannot take a tablespace offline immediately if the database is running in NOARCHIVELOG mode.</td>
</tr>
</tbody>
</table>

**Caution:** If you must take a tablespace offline, use the NORMAL clause (the default) if possible. This setting guarantees that the tablespace will not require recovery to come back online, even if after incomplete recovery you reset the redo log sequence using an ALTER DATABASE OPEN RESETLOGS statement.

Specify TEMPORARY only when you cannot take the tablespace offline normally. In this case, only the files taken offline because of errors need to be recovered before the tablespace can be brought online. Specify IMMEDIATE only after trying both the normal and temporary settings. The following example takes the users tablespace offline normally:

```sql
ALTER TABLESPACE users OFFLINE NORMAL;
```

### 8.8.3. Bringing Tablespaces Online

You can bring any tablespace in an Oracle Database online whenever the database is open. A tablespace is normally online so that the data contained within it is available to database users. If a tablespace to be brought online was not taken offline "cleanly" (that is, using the NORMAL clause of the ALTER TABLESPACE OFFLINE statement), you must first perform media recovery on the tablespace before bringing it online. Otherwise, the database returns an error and the tablespace remains offline.

### 8.8.4. Using Read-Only Tablespaces

Making a tablespace read-only prevents write operations on the datafiles in the tablespace. The primary purpose of read-only tablespaces is to eliminate the need to perform backup and recovery of large, static portions of a database. Read-only tablespaces also provide a way to protecting historical data so that users cannot modify it. Making a tablespace read-only prevents updates on all tables in the tablespace, regardless of a user's update privilege level.

**Note:** Making a tablespace read-only cannot in itself be used to satisfy archiving or data publishing requirements, because the tablespace can only be brought online in the database in which it was created. However, you can meet such requirements by using the transportable tablespace feature.

You can drop items, such as tables or indexes, from a read-only tablespace, but you cannot create or alter objects in a read-only tablespace. You can execute statements that update the file description in the data dictionary, such as ALTER TABLE...ADD or ALTER TABLE...MODIFY, but you will not be able to utilize the new description until the tablespace is made read/write. Read-only tablespaces can be transported to other databases. And, since read-only tablespaces can never be updated, they can reside on CD-ROM or WORM (Write Once-Read Many) devices.
8.8.5. Making a Tablespace Read-Only

- All tablespaces are initially created as read/write. Use the READ ONLY clause in the ALTER TABLESPACE statement to change a tablespace to read-only. You must have the ALTER TABLESPACE or MANAGE TABLESPACE system privilege. Before you can make a tablespace read-only, the following conditions must be met.
  - The tablespace must be online. This is necessary to ensure that there is no undo information that needs to be applied to the tablespace.
  - The tablespace cannot be the active undo tablespace or SYSTEM tablespace.
  - The tablespace must not currently be involved in an online backup, because the end of a backup updates the header file of all datafiles in the tablespace.

For better performance while accessing data in a read-only tablespace, you can issue a query that accesses all of the blocks of the tables in the tablespace just before making it read-only. A simple query, such as SELECT COUNT (*), executed against each table ensures that the data blocks in the tablespace can be subsequently accessed most efficiently. This eliminates the need for the database to check the status of the transactions that most recently modified the blocks. The following statement makes the flights tablespace read-only:

```
ALTER TABLESPACE flights READ ONLY;
```

You can issue the ALTER TABLESPACE...READ ONLY statement while the database is processing transactions. After the statement is issued, the tablespace is put into a transitional read-only state. No transactions are allowed to make further changes (using DML statements) to the tablespace. If a transaction attempts further changes, it is terminated and rolled back. However, transactions that already made changes and that attempt no further changes are allowed to commit or roll back. When there are transactions waiting to commit, the ALTER TABLESPACE...READ ONLY statement does not return immediately. It waits for all transactions started before you issued the ALTER TABLESPACE statement—even transactions that are against objects in different tablespaces—to either commit or roll back.

**Note:** This transitional read-only state only occurs if the value of the initialization parameter COMPATIBLE is 8.1.0 or greater. If this parameter is set to a value less than 8.1.0, the ALTER TABLESPACE...READ ONLY statement fails if any active transactions exist.

If you find it is taking a long time for the ALTER TABLESPACE statement to complete, you can identify the transactions that are preventing the read-only state from taking effect. You can then notify the owners of those transactions and decide whether to terminate the transactions, if necessary. The following example identifies the transaction entry for the ALTER TABLESPACE...READ ONLY statement and displays its session address (saddr):

```
SELECT SQL_TEXT, SADDR FROM V$SQLAREA, V$SESSION WHERE V$SQLAREA.ADDRESS = V$SESSION.SQL_ADDRESS AND SQL_TEXT LIKE 'alter tablespace%';
```

```
alter tablespace tbs1 read only         80034AF0
```

The start SCN of each active transaction is stored in the V$TRANSACTION view. Displaying this view sorted by ascending start SCN lists the transactions in execution order. From the preceding example, you already know the session address of the transaction entry for the read-only statement, and you can now locate it in the V$TRANSACTION view. All transactions with smaller start SCN, which indicates an earlier execution, can potentially hold up the quiesce and subsequent read-only state of the tablespace.

```
SELECT SES_ADDR, START_SCNB FROM V$TRANSACTION ORDER BY START_SCNB;
```

```
SES_ADDR  START_SCNB
----------- ----------
800352A0   3621   --> waiting on this txn
80035A50   3623   --> waiting on this txn
```
You can now find the owners of the blocking transactions.

```
SELECT T.SES_ADDR, S.USERNAME, S.MACHINE FROM V$SESSION S, V$TRANSACTION T
WHERE T.SES_ADDR = S.SADDR ORDER BY T.SES_ADDR
```

<table>
<thead>
<tr>
<th>SES_ADDR</th>
<th>USERNAME</th>
<th>MACHINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>800352A0</td>
<td>DAVIDB</td>
<td>DAVIDBLAP</td>
</tr>
<tr>
<td>80035A50</td>
<td>MIKEL</td>
<td>LAB61</td>
</tr>
<tr>
<td>80034AF0</td>
<td>DBA01</td>
<td>STEVEFLAP</td>
</tr>
</tbody>
</table>

After making the tablespace read-only, it is advisable to back it up immediately. As long as the tablespace remains read-only, no further backups of the tablespace are necessary, because no changes can be made to it.

### 8.8.6. Making a Read-Only Tablespace Writable

Use the `READ WRITE` keywords in the `ALTER TABLESPACE` statement to change a tablespace to allow write operations. You must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege. A prerequisite to making the tablespace read/write is that all of the datafiles in the tablespace, as well as the tablespace itself, must be online. Use the `DATAFILE...ONLINE` clause of the `ALTER DATABASE` statement to bring a datafile online. The `V$DATAFILE` view lists the current status of datafiles.

The following statement makes the flights tablespace writable:

```
ALTER TABLESPACE flights READ WRITE;
```

Making a read-only tablespace writable updates the control file entry for the datafiles, so that you can use the read-only version of the datafiles as a starting point for recovery.

### Altering a Bigfile Tablespace

Two clauses of the `ALTER TABLESPACE` statement support datafile transparency when you are using bigfile tablespaces:

- **RESIZE**: The `RESIZE` clause lets you resize the single datafile in a bigfile tablespace to an absolute size, without referring to the datafile. For example:

  ```
  ALTER TABLESPACE bigtbs RESIZE 80G;
  ```

- **AUTOEXTEND** (used outside of the `ADD DATAFILE` clause): With a bigfile tablespace, you can use the `AUTOEXTEND` clause outside of the `ADD DATAFILE` clause. For example:

  ```
  ALTER TABLESPACE bigtbs AUTOEXTEND ON NEXT 20G;
  ```

An error is raised if you specify an `ADD DATAFILE` clause for a bigfile tablespace.

### 8.7. Altering a Locally Managed Temporary Tablespace

Note: You cannot use the `ALTER TABLESPACE` statement, with the `TEMPORARY` keyword, to change a locally managed permanent tablespace into a locally managed temporary tablespace. You must use the `CREATE TEMPORARY TABLESPACE` statement to create a locally managed temporary tablespace.

You can use `ALTER TABLESPACE` to add a tempfile, take a tempfile offline, or bring a tempfile online, as illustrated in the following examples:

```
ALTER TABLESPACE lmtemp ADD TEMPFILE '/u02/oracle/data/lmtemp02.dbf' SIZE 18M REUSE;
ALTER TABLESPACE lmtemp TEMPFILE OFFLINE;
ALTER TABLESPACE lmtemp TEMPFILE ONLINE;
```
Note: You cannot take a temporary tablespace offline. Instead, you take its tempfile offline. The view V$TEMPFILE displays online status for a tempfile.

The ALTER DATABASE statement can be used to alter tempfiles. The following statements take offline and bring online tempfiles. They behave identically to the last two ALTER TABLESPACE statements in the previous example.

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' OFFLINE;
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' ONLINE;
```

The following statement resizes a tempfile:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' RESIZE 18M;
```

The following statement drops a tempfile and deletes its operating system file:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP INCLUDING DATAFILES;
```

The tablespace to which this tempfile belonged remains. A message is written to the alert log for the tempfile that was deleted. If an operating system error prevents the deletion of the file, the statement still succeeds, but a message describing the error is written to the alert log. It is also possible to use the ALTER DATABASE statement to enable or disable the automatic extension of an existing tempfile, and to rename (RENAME FILE) a tempfile.

Note: To rename a tempfile, you take the tempfile offline, use operating system commands to rename or relocate the tempfile, and then use the ALTER DATABASE RENAME FILE command to update the database controlfiles.

### 8.7.1. Shrinking a Locally Managed Temporary Tablespace

Large sort operations performed by the database may result in a temporary tablespace growing and occupying a considerable amount of disk space. After the sort operation completes, the extra space is not released; it is just marked as free and available for reuse. Therefore, a single large sort operation might result in a large amount of allocated temporary space that remains unused after the sort operation is complete. For this reason, the database enables you to shrink locally managed temporary tablespaces and release unused space.

You use the SHRINK SPACE clause of the ALTER TABLESPACE statement to shrink a temporary tablespace, or the SHRINK TEMPFILE clause of the ALTER TABLESPACE statement to shrink a specific tempfile of a temporary tablespace. Shrinking frees as much space as possible while maintaining the other attributes of the tablespace or tempfile. The optional KEEP clause defines a minimum size for the tablespace or tempfile. Shrinking is an online operation, which means that user sessions can continue to allocate sort extents if needed, and already-running queries are not affected. The following example shrinks the locally managed temporary tablespace lmtmp1 to a size of 20M.

```
ALTER TABLESPACE lmtmp1 SHRINK SPACE KEEP 20M;
```

The following example shrinks the tempfile lmtemp02.dbf of the locally managed temporary tablespace lmtmp2. Because the KEEP clause is omitted, the database attempts to shrink the tempfile to the minimum possible size.

```
ALTER TABLESPACE lmtmp2 SHRINK TEMPFILE '/u02/oracle/data/lmtemp02.dbf';
```

### Renaming Tablespaces

Using the RENAME TO clause of the ALTER TABLESPACE, you can rename a permanent or temporary tablespace. For example, the following statement renames the users tablespace:

```
ALTER TABLESPACE users RENAME TO usersts;
```
When you rename a tablespace the database updates all references to the tablespace name in the data dictionary, control file, and (online) datafile headers. The database does not change the tablespace ID so if this tablespace were, for example, the default tablespace for a user, then the renamed tablespace would show as the default tablespace for the user in the DBA_USERS view.

The following affect the operation of this statement:

- The COMPATIBLE parameter must be set to 10.0.0 or higher. If the tablespace being renamed is the SYSTEM tablespace or the SYSAUX tablespace, then it will not be renamed and an error is raised.
- If any datafile in the tablespace is offline, or if the tablespace is offline, then the tablespace is not renamed and an error is raised.
- If the tablespace is read only, then datafile headers are not updated. This should not be regarded as corruption; instead, it causes a message to be written to the alert log indicating that datafile headers have not been renamed. The data dictionary and control file are updated.
- If the tablespace is the default temporary tablespace, then the corresponding entry in the database properties table is updated and the DATABASE_PROPERTIES view shows the new name.
- If the tablespace is an undo tablespace and if the following conditions are met, then the tablespace name is changed to the new tablespace name in the server parameter file (SPFILE).
  - The server parameter file was used to start up the database.
  - The tablespace name is specified as the UNDO_TABLESPACE for any instance.
- If a traditional initialization parameter file (PIFILE) is being used then a message is written to the alert log stating that the initialization parameter file must be manually changed.

8.8. Dropping Tablespaces

You can drop a tablespace and its contents (the segments contained in the tablespace) from the database if the tablespace and its contents are no longer required. You must have the DROP TABLESPACE system privilege to drop a tablespace.

Caution: Once a tablespace has been dropped, the data in the tablespace is not recoverable. Therefore, make sure that all data contained in a tablespace to be dropped will not be required in the future. Also, immediately before and after dropping a tablespace from a database, back up the database completely. This is strongly recommended so that you can recover the database if you mistakenly drop a tablespace, or if the database experiences a problem in the future after the tablespace has been dropped.

When you drop a tablespace, the file pointers in the control file of the associated database are removed. You can optionally direct Oracle Database to delete the operating system files (datafiles) that constituted the dropped tablespace. If you do not direct the database to delete the datafiles at the same time that it deletes the tablespace, you must later use the appropriate commands of your operating system to delete them. You cannot drop a tablespace that contains any active segments. For example, if a table in the tablespace is currently being used or the tablespace contains undo data needed to roll back uncommitted transactions, you cannot drop the tablespace. The tablespace can be online or offline, but it is best to take the tablespace offline before dropping it. To drop a tablespace, use the DROP TABLESPACE statement. The following statement drops the users tablespace, including the segments in the tablespace:

```
DROP TABLESPACE users INCLUDING CONTENTS;
```

If the tablespace is empty (does not contain any tables, views, or other structures), you do not need to specify the INCLUDING CONTENTS clause. Use the CASCADE CONSTRAINTS clause to drop all referential integrity constraints from tables outside the tablespace that refer to primary and unique keys of tables inside the tablespace. To delete the datafiles associated with a tablespace at the same time that the tablespace is dropped, use the INCLUDING CONTENTS AND DATAFILES clause. The following statement drops the users tablespace and its associated datafiles:

```
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;
```
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;

A message is written to the alert log for each datafile that is deleted. If an operating system error prevents the deletion of a file, the DROP TABLESPACE statement still succeeds, but a message describing the error is written to the alert log.
9. Transportable Tablespace

9.1. Introduction to Transportable Tablespaces

You can use the Transportable Tablespace feature to copy a set of tablespace(s) from one Oracle Database to another.

**Note:** This method for transporting tablespaces requires that you place the tablespaces to be transported in read-only mode until you complete the transporting process. If this is undesirable, you can use the Transportable Tablespace from Backup feature.

The tablespaces being transported can be either dictionary managed or locally managed. Starting with Oracle9i, the transported tablespaces are not required to be of the same block size as the target database standard block size. Moving data using transportable tablespaces is much faster than performing either an export/import or unload/load of the same data. This is because the datafiles containing all of the actual data are just copied to the destination location, and you use Data Pump to transfer only the metadata of the tablespace objects to the new database.

9.2. Transporting Tablespaces between Databases

**Note:** You must be using the Enterprise Edition of Oracle8i or later to generate a transportable tablespace set. However, you can use any edition of Oracle8i or later to import a transportable tablespace set into an Oracle Database on the same platform. To import a transportable tablespace set into an Oracle Database on a different platform, both databases must have compatibility set to at least 10.0.

9.3. About Transporting Tablespaces across Platforms

Starting with Oracle Database 11g, you can transport tablespaces across platforms. This functionality can be used to:

- Allow a database to be migrated from one platform to another
- Provide an easier and more efficient means for content providers to publish structured data and distribute it to customers running Oracle Database on different platforms
- Simplify the distribution of data from a data warehouse environment to data marts, which are often running on smaller platforms
- Enable the sharing of read-only tablespaces between Oracle Database installations on different operating systems or platforms, assuming that your storage system is accessible from those platforms and the platforms all have the same endianness, as described in the sections that follow
- Many, but not all, platforms are supported for cross-platform tablespace transport.

You can query the V$TRANSPORTABLE_PLATFORM view to see the platforms that are supported, and to determine each platform’s endian format (byte ordering). The following query displays the platforms that support cross-platform tablespace transport:

```
SQL> COLUMN PLATFORM_NAME FORMAT A32
SQL> SELECT * FROM V$TRANSPORTABLE_PLATFORM;
```

<table>
<thead>
<tr>
<th>PLATFORM_ID</th>
<th>PLATFORM_NAME</th>
<th>ENDIAN_FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Solaris[tm] OE (32-bit)</td>
<td>Big</td>
</tr>
<tr>
<td>2</td>
<td>Solaris[tm] OE (64-bit)</td>
<td>Big</td>
</tr>
<tr>
<td>7</td>
<td>Microsoft Windows IA (32-bit)</td>
<td>Little</td>
</tr>
<tr>
<td>10</td>
<td>Linux IA (32-bit)</td>
<td>Little</td>
</tr>
<tr>
<td>6</td>
<td>AIX-Based Systems (64-bit)</td>
<td>Big</td>
</tr>
</tbody>
</table>
If the source platform and the target platform are of different endianness, then an additional step must be done on either the source or target platform to convert the tablespace being transported to the target format. If they are of the same endianness, then no conversion is necessary and tablespaces can be transported as if they were on the same platform. Before a tablespace can be transported to a different platform, the datafile header must identify the platform to which it belongs. In an Oracle Database with compatibility set to 10.0.0 or later, you can accomplish this by making the datafile read/write at least once.

9.4. Limitations on Transportable Tablespace Use

Be aware of the following limitations as you plan to transport tablespaces:

- The source and target database must use the same character set and national character set.
- You cannot transport a tablespace to a target database in which a tablespace with the same name already exists. However, you can rename either the tablespace to be transported or the destination tablespace before the transport operation.
- Objects with underlying objects (such as materialized views) or contained objects (such as partitioned tables) are not transportable unless all of the underlying or contained objects are in the tablespace set.

Encrypted tablespaces have the following the limitations:

- Before transporting an encrypted tablespace, you must copy the Oracle wallet manually to the destination database, unless the master encryption key is stored in a Hardware Security Module (HSM) device instead of an Oracle wallet. When copying the wallet, the wallet password remains the same in the destination database. However, it is recommended that you change the password on the destination database so that each database has its own wallet password.
- You cannot transport an encrypted tablespace to a database that already has an Oracle wallet for transparent data encryption. In this case, you must use Oracle Data Pump to export the tablespace’s schema objects and then import them to the destination database. You can optionally take advantage of Oracle Data Pump features that enable you to maintain encryption for the data while it is being exported and imported.
- You cannot transport an encrypted tablespace to a platform with different endianness.
- Tablespaces that do not use block encryption but that contain tables with encrypted columns cannot be transported. You must use Oracle Data Pump to export and import the tablespace’s schema objects. You can take advantage of Oracle Data Pump features that enable you to maintain encryption for the data while it is being exported and imported.
• SYSTEM Tablespace Objects You cannot transport the SYSTEM tablespace or objects owned by the user SYS. Some examples of such objects are PL/SQL, Java classes, callouts, views, synonyms, users, privileges, dimensions, directories, and sequences.

9.5. Transport Scenario Source Database Target Database

• Databases on the same platform 8.0 8.0
• Tablespace with different database block 9.0 9.0 size than the target database
• Databases on different platforms 10.0 10.0

Transporting Tablespaces between Databases: A Procedure and Example
The following steps summarize the process of transporting a tablespace. Details for each step are provided in the subsequent example.
1. For cross-platform transport, check the endian format of both platforms by querying the V$TRANSPORTABLE_PLATFORM view. Ignore this step if you are transporting your tablespace set to the same platform.
2. Pick a self-contained set of tablespaces.
3. Generate a transportable tablespace set.
A transportable tablespace set (or transportable set) consists of datafiles for the set of tablespaces being transported and an export file containing structural information (metadata) for the set of tablespaces. You use Data Pump to perform the export.
If you are transporting the tablespace set to a platform with different endianness from the source platform, you must convert the tablespace set to the endianness of the target platform. You can perform a source-side conversion at this step in the procedure, or you can perform a target-side conversion as part of step 4.

Note: This method of generating a transportable tablespace requires that you temporarily make the tablespace read-only. If this is undesirable, you can use the alternate method known as transportable tablespace from backup.

4. Transport the tablespace set.
Copy the datafiles and the export file to a place that is accessible to the target database.
If you have transported the tablespace set to a platform with different endianness from the source platform, and you have not performed a source-side conversion to the endianness of the target platform, you should perform a target-side conversion now.
5. Import the tablespace set.
Invoke the Data Pump utility to import the metadata for the set of tablespaces into the target database.

Example
The steps for transporting a tablespace are illustrated more fully in the example that follows, where it is assumed the following datafiles and tablespaces exist:

<table>
<thead>
<tr>
<th>Tablespace</th>
<th>Datafile</th>
</tr>
</thead>
<tbody>
<tr>
<td>sales_1</td>
<td>/u01/oracle/oradata/salesdb/sales_101.dbf</td>
</tr>
<tr>
<td>sales_2</td>
<td>/u01/oracle/oradata/salesdb/sales_201.dbf</td>
</tr>
</tbody>
</table>

Step 1: Determine if Platforms are Supported and Determine Endianness
This step is only necessary if you are transporting the tablespace set to a platform different from the source platform. If you are transporting the tablespace set to a platform different from the source platform, then determine if cross-platform tablespace transport is supported for both the source and target platforms, and determine the endianness of each platform. If both platforms have the same endianness, no conversion is necessary. Otherwise you must do a conversion of the tablespace set either at the source or target database. If you are transporting sales_1 and sales_2 to a different platform, you can execute the following query on each platform. If the query returns a row, the platform supports cross-platform tablespace transport.

```
SELECT d.PLATFORM_NAME, ENDIAN_FORMAT FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
```
WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME;

The following is the query result from the source platform:

<table>
<thead>
<tr>
<th>PLATFORM_NAME</th>
<th>ENDIAN_FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris[tm] OE (32-bit)</td>
<td>Big</td>
</tr>
</tbody>
</table>

The following is the result from the target platform:

<table>
<thead>
<tr>
<th>PLATFORM_NAME</th>
<th>ENDIAN_FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows NT</td>
<td>Little</td>
</tr>
</tbody>
</table>

You can see that the endian formats are different and thus a conversion is necessary for transporting the tablespace set.

Step 2: Pick a Self-Contained Set of Tablespaces

There may be logical or physical dependencies between objects in the transportable set and those outside of the set. You can only transport a set of tablespaces that is self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the tablespaces. Some examples of self contained tablespace violations are: An index inside the set of tablespaces is for a table outside of the set of tablespaces.

Note: It is not a violation if a corresponding index for a table is outside of the set of tablespaces.

- A partitioned table is partially contained in the set of tablespaces.
- The tablespace set you want to copy must contain either all partitions of a partitioned table, or none of the partitions of a partitioned table. If you want to transport a subset of a partition table, you must exchange the partitions into tables.
- A referential integrity constraint points to a table across a set boundary. When transporting a set of tablespaces, you can choose to include referential integrity constraints. However, doing so can affect whether or not a set of tablespaces is self-contained. If you decide not to transport constraints, then the constraints are not considered as pointers. A table inside the set of tablespaces contains a LOB column that points to LOBs outside the set of tablespaces.

When you invoke the DBMS_TTS package, you specify the list of tablespaces in the transportable set to be checked for self containment. You can optionally specify if constraints must be included. For strict or full containment, you must additionally set the TTS_FULL_CHECK parameter to TRUE. The strict or full containment check is for cases that require capturing not only references going outside the transportable set, but also those coming into the set. Tablespace Point-in-Time Recovery (TSPITR) is one such case where dependent objects must be fully contained or fully outside the transportable set. For example, it is a violation to perform TSPITR on a tablespace containing a table t but not its index i because the index and data will be inconsistent after the transport. A full containment check ensures that there are no dependencies going outside or coming into the transportable set.

Note: The default for transportable tablespaces is to check for self containment rather than full containment.

The following statement can be used to determine whether tablespaces sales_1 and sales_2 are self-contained, with referential integrity constraints taken into consideration (indicated by TRUE).

EXECUTE DBMS_TTS.TRANSPORT_SET_CHECK ('sales_1, sales_2', TRUE);

After invoking this PL/SQL package, you can see all violations by selecting from the TRANSPORT_SET_VIOLATIONS view. If the set of tablespaces is self-contained, this view is empty. The following example illustrates a case where there are two violations: a foreign key constraint, dept_fk, across the tablespace set boundary, and a partitioned table, Jim. sales, that is partially contained in the tablespace set.
Oracle 11g - Transportable Tablespace

SQL> SELECT * FROM TRANSPORT_SET_VIOLATIONS;

VIOLATIONS
---------------------------------------------------------------------------
Constraint DEPT_FK between table JIM.EMP in tablespace SALES_1 and table JIM.DEPT in tablespace OTHER Partitioned table JIM.SALES is partially contained in the transportable set

These violations must be resolved before sales_1 and sales_2 are transportable. As noted in the next step, one choice for bypassing the integrity constraint violation is to not export the integrity constraints.

Step 3: Generate a Transportable Tablespace Set
Any privileged user can perform this step. However, you must have been assigned the EXP_FULL_DATABASE role to perform a transportable tablespace export operation.

Note: This method of generating a transportable tablespace requires that you temporarily make the tablespace read-only. If this is undesirable, you can use the alternate method known as transportable tablespace from backup.

After ensuring you have a self-contained set of tablespaces that you want to transport, generate a transportable tablespace set by performing the following actions:

1. Make all tablespaces in the set you are copying read-only.
   SQL> ALTER TABLESPACE sales_1 READ ONLY;
   Tablespace altered.
   SQL> ALTER TABLESPACE sales_2 READ ONLY;
   Tablespace altered.

2. Invoke the Data Pump export utility on the host system and specify which tablespaces are in the transportable set.
   SQL> HOST
   $ EXPDP system/password DUMPFILE=expdat.dmp DIRECTORY=dpump_dir
   TRANSPORT_TABLESPACES = sales_1,sales_2

   • You must always specify TRANSPORT_TABLESPACES, which determines the mode of the export operation. In this example:

   • The DUMPFILE parameter specifies the name of the structural information export file to be created, expdat.dmp.

   • The DIRECTORY parameter specifies the default directory object that points to the operating system or Automatic Storage Management location of the dump file. You must create the DIRECTORY object before invoking Data Pump, and you must grant the READ and WRITE object privileges on the directory to PUBLIC.

   • Triggers and indexes are included in the export operation by default.

If you want to perform a transport tablespace operation with a strict containment check, use the TRANSPORT_FULL_CHECK parameter, as shown in the following example:

EXPDP system/password DUMPFILE=expdat.dmp DIRECTORY = dpump_dir
   TRANSPORT_TABLESPACES=sales_1,sales_2 TRANSPORT_FULL_CHECK=Y

In this example, the Data Pump export utility verifies that there are no dependencies between the objects inside the transportable set and objects outside the transportable set. If the tablespace set being transported is not self-contained, then the export fails and indicates that the transportable set is not self-contained. You must then return to Step 1 to resolve all violations.
Note: The Data Pump utility is used to export only data dictionary structural information (metadata) for the tablespaces. No actual data is unloaded, so this operation goes relatively quickly even for large tablespace sets.

3. When finished, exit back to SQL*Plus:

   $ EXIT

If sales_1 and sales_2 are being transported to a different platform, and the endianness of the platforms is different, and if you want to convert before transporting the tablespace set, then convert the datafiles composing the sales_1 and sales_2 tablespaces:

4. From SQL*Plus, return to the host system:

   SQL> HOST

5. The RMAN CONVERT command is used to do the conversion. Start RMAN and connect to the target database:

   $ RMAN TARGET /

   Recovery Manager: Release 10.1.0.0.0
   Copyright (c) 1995, 2003, Oracle Corporation. All rights reserved.
   connected to target database: salesdb (DBID=3295731590)

6. Convert the datafiles into a temporary location on the source platform. In this example, assume that the temporary location, directory /temp, has already been created. The converted datafiles are assigned names by the system.

   RMAN> CONVERT TABLESPACE sales_1,sales_2
   2> TO PLATFORM 'Microsoft Windows NT'
   3> FORMAT '/temp/%U';
   Starting backup at 08-APR-03
   using target database control file instead of recovery catalog
   allocated channel: ORA_DISK_1
   channel ORA_DISK_1: sid=11 devtype=DISK
   channel ORA_DISK_1: starting datafile conversion
   input datafile fno=00005 name=/u01/oracle/oradata/salesdb/sales_101.dbf
   converted datafile=/temp/data_D-10_I-3295731590_TS-ADMIN_TBS_FNO-5_05ek24v5
   channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:15
   channel ORA_DISK_1: starting datafile conversion
   input datafile fno=00004 name=/u01/oracle/oradata/salesdb/sales_101.dbf
   converted datafile=/temp/data_D-10_I-3295731590_TS-EXAMPLE_FNO-4_06ek24vl
   channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:45
   Finished backup at 08-APR-03

7. Exit Recovery Manager:

   RMAN> exit
   Recovery Manager complete.

Step 4: Transport the Tablespace Set

Transport both the datafiles and the export file of the tablespaces to a place that is accessible to the target database. If both the source and destination are file systems, you can use:

- Any facility for copying flat files (for example, an operating system copy utility or ftp)
- The DBMS_FILE_TRANSFER package
- RMAN
- Any facility for publishing on CDs
- If either the source or destination is an Automatic Storage Management (ASM) disk group, you can use:ftp to or from the /sys/asm virtual folder in the XML DB repository
**Caution:** Exercise caution when using the UNIX `dd` utility to copy raw-device files between databases. The `dd` utility can be used to copy an entire source raw-device file, or it can be invoked with options that instruct it to copy only a specific range of blocks from the source raw-device file.

It is difficult to ascertain actual datafile size for a raw-device file because of hidden control information that is stored as part of the datafile. Thus, it is advisable when using the `dd` utility to specify copying the entire source raw-device file contents. If you are transporting the tablespace set to a platform with endianness that is different from the source platform, and you have not yet converted the tablespace set, you must do so now. This example assumes that you have completed the following steps before the transport:

1. Set the source tablespaces to be transported to be read-only.
2. Use the export utility to create an export file (in our example, `expdat.dmp`). Datafiles that are to be converted on the target platform can be moved to a temporary location on the target platform. However, all datafiles, whether already converted or not, must be moved to a designated location on the target database.

Now use RMAN to convert the necessary transported datafiles to the endian format of the destination host format and deposit the results in `/orahome/dbs`, as shown in this hypothetical example:

```sql
RMAN> CONVERT DATAFILE
2> '/hq/finance/work/tru/tbs_31.f',
3> '/hq/finance/work/tru/tbs_32.f',
4> '/hq/finance/work/tru/tbs_41.f'
5> TO PLATFORM=“Solaris[tm] OE (32-bit)”
6> FROM PLATFORM=“HP TRu64 UNIX”
7> DB_FILE_NAME_CONVERT=
8> "'/hq/finance/work/tru/", "/hq/finance/dbs/tru"
9> PARALLELISM=5;
```

You identify the datafiles by filename, not by tablespace name. Until the tablespace metadata is imported, the local instance has no way of knowing the desired tablespace names. The source and destination platforms are optional. RMAN determines the source platform by examining the datafile, and the target platform defaults to the platform of the host running the conversion.

**Step 5: Import the Tablespace Set**

**Note:** If you are transporting a tablespace of a different block size than the standard block size of the database receiving the tablespace set, then you must first have a `DB_nK_CACHE_SIZE` initialization parameter entry in the receiving database parameter file.

For example, if you are transporting a tablespace with an 8K block size into a database with a 4K standard block size, then you must include a `DB_8K_CACHE_SIZE` initialization parameter entry in the parameter file. If it is not already included in the parameter file, this parameter can be set using the `ALTER SYSTEM SET` statement.

Any privileged user can perform this step. To import a tablespace set, perform the following tasks:

1. Import the tablespace metadata using the Data Pump Import utility, `impdp`:

   ```sql
   IMPDP system/password DUMPFILE=expdat.dmp DIRECTORY=dpump_dir
   TRANSPORT_DATAFILES= /salesdb/sales_101.dbf,
   /salesdb/sales_201.dbf
   REMAP_SCHEMA=(dcranney:smith) REMAP_SCHEMA=(jfee:williams)
   ```

   In this example we specify the following:

   - The `DUMPFILE` parameter specifies the exported file containing the metadata for the tablespaces to be imported.
   - The `DIRECTORY` parameter specifies the directory object that identifies the location of the dump file.
• The TRANSPORT_DATAFILES parameter identifies all of the datafiles containing the tablespaces to be imported.

The REMAP_SCHEMA parameter changes the ownership of database objects. If you do not specify REMAP_SCHEMA, all database objects (such as tables and indexes) are created in the same user schema as in the source database, and those users must already exist in the target database. If they do not exist, then the import utility returns an error. In this example, objects in the tablespace set owned by dcranney in the source database will be owned by smith in the target database after the tablespace set is imported. Similarly, objects owned by jfee in the source database will be owned by williams in the target database. In this case, the target database is not required to have users dcranney and jfee, but must have users smith and williams. After this statement executes successfully, all tablespaces in the set being copied remain in read-only mode. Check the import logs to ensure that no error have occurred. When dealing with a large number of datafiles, specifying the list of datafile names in the statement line can be a laborious process. It can even exceed the statement line limit. In this situation, you can use an import parameter file. For example, you can invoke the Data Pump import utility as follows:

```
IMPDP system/password PARFILE='par.f'
```

Where the parameter file, par.f contains the following:

```
DIRECTORY=dpump_dir
DUMPFILE=expdat.dmp
TRANSPORT_DATAFILES="'/db/sales_jan','/db/sales_feb'"
REMAP_SCHEMA=dcranney:smith
REMAP_SCHEMA=jfee:williams
```

2. If required, put the tablespaces into read/write mode as follows:

```
ALTER TABLESPACE sales_1 READ WRITE;
ALTER TABLESPACE sales_2 READ WRITE;
```

### 9.6. Guidelines for Managing Datafiles

Datafiles are physical files of the operating system that store the data of all logical structures in the database. They must be explicitly created for each tablespace.

**Note:** Tempfiles are a special class of data files that are associated only with temporary tablespaces. Information in this chapter applies to both datafiles and tempfiles except where differences are noted.

Oracle Database assigns each datafile two associated file numbers, an absolute file number and a relative file number, that are used to uniquely identify it. These numbers are described in the following table:

#### 9.6.1. Type of File Number Description

Absolute uniquely identifies a datafile in the database. This file number can be used in many SQL statements that reference datafiles in place of using the file name. The absolute file number can be found in the FILE# column of the V$DATAFILE or V$TEMPFILE view, or in the FILE_ID column of the DBA_DATA_FILES or DBA_TEMP_FILES view. Relative uniquely identifies a datafile within a tablespace. For small and medium size databases, relative file numbers usually have the same value as the absolute file number. However, when the number of datafiles in a database exceeds a threshold (typically 1023), the relative file number differs from the absolute file number. In a bigfile tablespace, the relative file number is always 1024 (4096 on OS/390 platform).

#### 9.6.2. Determine the Number of Datafiles

At least one datafile is required for the SYSTEM and SYSAUX tablespaces of a database. Your database should contain several other tablespaces with their associated datafiles or tempfiles. The number of datafiles that you anticipate creating for your database can affect the settings of initialization parameters and the specification of CREATE DATABASE statement clauses. Be aware that your operating system might impose limits on the number of datafiles contained in your Oracle Database. Also consider that the number of datafiles, and how and where they are allocated can affect the performance of your database.
Note: One means of controlling the number of datafiles in your database and simplifying their management is to use bigfile tablespaces. Bigfile tablespaces comprise a single, very large datafile and are especially useful in ultra large databases and where a logical volume manager is used for managing operating system files.

Consider the following guidelines when determining the number of datafiles for your database.

9.6.3. Determine a Value for the DB_FILES Initialization Parameter

When starting an Oracle Database instance, the DB_FILES initialization parameter indicates the amount of SGA space to reserve for datafile information and thus, the maximum number of datafiles that can be created for the instance. This limit applies for the life of the instance. You can change the value of DB_FILES (by changing the initialization parameter setting), but the new value does not take effect until you shut down and restart the instance.

When determining a value for DB_FILES, take the following into consideration:

- If the value of DB_FILES is too low, you cannot add datafiles beyond the DB_FILES limit without first shutting down the database.
- If the value of DB_FILES is too high, memory is unnecessarily consumed.
- Consider Possible Limitations When Adding Datafiles to a Tablespace

You can add datafiles to traditional smallfile tablespaces, subject to the following limitations:

- Operating systems often impose a limit on the number of files a process can open simultaneously. More datafiles cannot be created when the operating system limit of open files is reached.
- Operating systems impose limits on the number and size of datafiles.
- The database imposes a maximum limit on the number of datafiles for any Oracle Database opened by any instance. This limit is operating system specific.
- You cannot exceed the number of datafiles specified by the DB_FILES initialization parameter.
- When you issue CREATE DATABASE or CREATE CONTROLFILE statements, the MAXDATAFILES parameter specifies an initial size of the datafile portion of the control file. However, if you attempt to add a new file whose number is greater than MAXDATAFILES, but less than or equal to DB_FILES, the control file will expand automatically so that the datafiles section can accommodate more files.

9.6.4. Consider the Performance Impact

The number of datafiles contained in a tablespace, and ultimately the database, can have an impact upon performance. Oracle Database allows more datafiles in the database than the operating system defined limit. The database DBWN processes can open all online datafiles. Oracle Database is capable of treating open file descriptors as a cache, automatically closing files when the number of open file descriptors reaches the operating system-defined limit. This can have a negative performance impact. When possible, adjust the operating system limit on open file descriptors so that it is larger than the number of online data files in the database.

9.6.5. Determine the Size of Datafiles

When creating a tablespace, you should estimate the potential size of database objects and create sufficient datafiles. Later, if needed, you can create additional datafiles and add them to a tablespace to increase the total amount of disk space allocated to it, and consequently the database. Preferably, place datafiles on multiple devices to ensure that data is spread evenly across all devices.

9.6.6. Store Datafiles Separate from Redo Log Files

Datafiles should not be stored on the same disk drive that stores the database redo log files. If the datafiles and redo log files are stored on the same disk drive and that disk drive fails, the files cannot be used in your database recovery procedures. If you multiplex your redo log files, then the likelihood of losing all of your redo log files is low,
so you can store datafiles on the same drive as some redo log files.

9.6.7. Creating Datafiles and Adding Datafiles to a Tablespace

You can create datafiles and associate them with a tablespace using any of the statements listed in the following table. In all cases, you can either specify the file specifications for the datafiles being created, or you can use the Oracle-managed files feature to create files that are created and managed by the database server. If you add new datafiles to a tablespace and do not fully specify the filenames, the database creates the datafiles in the default database directory or the current directory, depending upon your operating system. Oracle recommends you always specify a fully qualified name for a datafile. Unless you want to reuse existing files, make sure the new filenames do not conflict with other files. Old files that have been previously dropped will be overwritten. If a statement that creates a datafile fails, the database removes any created operating system files. However, because of the large number of potential errors that can occur with file systems and storage subsystems, there can be situations where you must manually remove the files using operating system commands.

9.6.8. Changing Datafile Size

- Enabling and Disabling Automatic Extension for a Datafile

You can create datafiles or alter existing datafiles so that they automatically increase in size when more space is needed in the database. The file size increases in specified increments up to a specified maximum.

- Setting your datafiles to extend automatically provides these advantages:
  - Reduces the need for immediate intervention when a tablespace runs out of space
  - Ensures applications will not halt or be suspended because of failures to allocate extents

To determine whether a datafile is auto-extensible, query the DBA_DATA_FILES view and examine the AUTOEXTENSIBLE column.

You can specify automatic file extension by specifying an AUTOEXTEND ON clause when you create datafiles using the following SQL statements:

- CREATE DATABASE
- ALTER DATABASE
- CREATE TABLESPACE
- ALTER TABLESPACE

You can enable or disable automatic file extension for existing datafiles, or manually resize a datafile, using the ALTER DATABASE statement. For a bigfile tablespace, you are able to perform these operations using the ALTER TABLESPACE statement. The following example enables automatic extension for a datafile added to the users tablespace:

```
ALTER TABLESPACE users ADD DATAFILE '/u02/oracle/rdbb1/users03.dbf' SIZE 10M AUTOEXTEND ON NEXT 512K MAXSIZE 250M;
```

The value of NEXT is the minimum size of the increments added to the file when it extends. The value of MAXSIZE is the maximum size to which the file can automatically extend. The next example disables the automatic extension for the datafile.

```
ALTER DATABASE DATAFILE '/u02/oracle/rdbb1/users03.dbf' AUTOEXTEND OFF;
```

9.6.9. Manually Resizing a Datafile

You can manually increase or decrease the size of a datafile using the ALTER DATABASE statement. This enables you to add more space to your database without adding more datafiles. This is beneficial if you are concerned about reaching the maximum number of datafiles allowed in your database. For a bigfile tablespace you can use the ALTER
TABLESPACE statement to resize a datafile. You are not allowed to add a datafile to a bigfile tablespace. Manually reducing the sizes of datafiles enables you to reclaim unused space in the database. This is useful for correcting errors in estimates of space requirements. In the next example, assume that the datafile /u02/oracle/rbdb1/stuff01.dbf has extended up to 250M. However, because its tablespace now stores smaller objects, the datafile can be reduced in size. The following statement decreases the size of datafile /u02/oracle/rbdb1/stuff01.dbf:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf'
    RESIZE 100M;
```

**Note:** It is not always possible to decrease the size of a file to a specific value. It could be that the file contains data beyond the specified decreased size, in which case the database will return an error.

### 9.6.10. Altering Datafile Availability

You can alter the availability of individual datafiles or tempfiles by taking them offline or bringing them online. Offline datafiles are unavailable to the database and cannot be accessed until they are brought back online. Reasons for altering datafile availability include the following:

- You want to perform an offline backup of a datafile.
- You want to rename or relocate a datafile. You must first take it offline or take the tablespace offline.
- The database has problems writing to a datafile and automatically takes the datafile offline. Later, after resolving the problem, you can bring the datafile back online manually.
- A datafile becomes missing or corrupted. You must take it offline before you can open the database.

The datafiles of a read-only tablespace can be taken offline or brought online, but bringing a file online does not affect the read-only status of the tablespace. You cannot write to the datafile until the tablespace is returned to the read/write state.

**Note:** You can make all datafiles of a tablespace temporarily unavailable by taking the tablespace itself offline. You must leave these files in the tablespace to bring the tablespace back online, although you can relocate or rename them following procedures.

To take a datafile offline or bring it online, you must have the ALTER DATABASE system privilege. To take all datafiles or tempfiles offline using the ALTER TABLESPACE statement, you must have the ALTER TABLESPACE or MANAGE TABLESPACE system privilege. In an Oracle Real Application Clusters environment, the database must be open in exclusive mode. This section describes ways to alter datafile availability, and contains the following topics:

- Bringing Datafiles Online or Taking Offline in ARCHIVELOG Mode
- Taking Datafiles Offline in NOARCHIVELOG Mode
- Altering the Availability of All Datafiles or Tempfiles in a Tablespace

### 9.7. Bringing Datafiles Online or Taking Offline in ALM

To bring an individual datafile online, issue the ALTER DATABASE statement and include the DATAFILE clause. The following statement brings the specified datafile online:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' ONLINE;
```

To take the same file offline, issue the following statement:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' OFFLINE;
```

**Note:** To use this form of the ALTER DATABASE statement, the database must be in ARCHIVELOG mode. This requirement prevents you from accidentally losing the datafile, since taking the datafile offline while in
NOARCHIVELOG mode is likely to result in losing the file.

9.8. Taking Datafiles Offline in NOARCHIVELOG Mode

To take a datafile offline when the database is in NOARCHIVELOG mode, use the ALTER DATABASE statement with both the DATAFILE and OFFLINE FOR DROP clauses.

- The OFFLINE keyword causes the database to mark the datafile OFFLINE, whether or not it is corrupted, so that you can open the database.
- The FOR DROP keywords mark the datafile for subsequent dropping. Such a datafile can no longer be brought back online.

Note: This operation does not actually drop the datafile. It remains in the data dictionary, and you must drop it yourself using one of the following methods:

- An ALTER TABLESPACE ... DROP DATAFILE statement. After an OFFLINE FOR DROP, this method works for dictionary managed tablespaces only.
- A DROP TABLESPACE ... INCLUDING CONTENTS AND DATAFILES statement
- If the preceding methods fail, an operating system command to delete the datafile. This is the least desirable method, as it leaves references to the datafile in the data dictionary and control files.

The following statement takes the specified datafile offline and marks it to be dropped:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf' OFFLINE FOR DROP;
```

9.9. Altering the Availability of All Datafiles or Tempfiles in a Tablespace

Clauses of the ALTER TABLESPACE statement allow you to change the online or offline status of all of the datafiles or tempfiles within a tablespace. Specifically, the statements that affect online/offline status are:

- ALTER TABLESPACE ... DATAFILE {ONLINE|OFFLINE}
- ALTER TABLESPACE ... TEMPFILE {ONLINE|OFFLINE}

You are required only to enter the tablespace name, not the individual datafiles or tempfiles. All of the datafiles or tempfiles are affected, but the online/offline status of the tablespace itself is not changed. In most cases the preceding ALTER TABLESPACE statements can be issued whenever the database is mounted, even if it is not open. However, the database must not be open if the tablespace is the SYSTEM tablespace, an undo tablespace, or the default temporary tablespace. The ALTER DATABASE DATAFILE and ALTER DATABASE TEMPFILE statements also have ONLINE/OFFLINE clauses; however in those statements you must enter all of the filenames for the tablespace. The syntax is different from the ALTER TABLESPACE...ONLINE/OFFLINE statement that alters tablespace availability, because that is a different operation. The ALTER TABLESPACE statement takes datafiles offline as well as the tablespace, but it cannot be used to alter the status of a temporary tablespace or its tempfile(s).

9.10. Renaming and Relocating Datafiles

You can rename datafiles to either change their names or relocate them. When you rename and relocate datafiles with these procedures, only the pointers to the datafiles, as recorded in the database control file, are changed. The procedures do not physically rename any operating system files, nor do they copy files at the operating system level. Renaming and relocating datafiles involves several steps. Read the steps and examples carefully before performing these procedures.

Procedures for Renaming and Relocating Datafiles in a Single Tablespace

The section suggests some procedures for renaming and relocating datafiles that can be used for a single
tablespace. You must have ALTER TABLESPACE system privileges.

**Procedure for Renaming Datafiles in a Single Tablespace**

To rename datafiles in a single tablespace, complete the following steps:

1. Take the tablespace that contains the datafiles offline. The database must be open.

For example:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

2. Rename the datafiles using the operating system.

3. Use the ALTER TABLESPACE statement with the RENAME DATAFILE clause to change the filenames within the database. For example, the following statement renames the datafiles `/u02/oracle/rbdb1/user1.dbf` and `/u02/oracle/rbdb1/user2.dbf` to `/u02/oracle/rbdb1/users01.dbf` and `/u02/oracle/rbdb1/users02.dbf`, respectively:

```
ALTER TABLESPACE users RENAME DATAFILE '/u02/oracle/rbdb1/user1.dbf',
'/u02/oracle/rbdb1/user2.dbf' TO '/u02/oracle/rbdb1/users01.dbf',
'/u02/oracle/rbdb1/users02.dbf';
```

Always provide complete filenames (including their paths) to properly identify the old and new datafiles. In particular, specify the old datafile name exactly as it appears in the DBA_DATAFILES view of the data dictionary.

4. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

**Procedure for Relocating Datafiles in a Single Tablespace**

Here is a sample procedure for relocating a datafile. Assume the following conditions:

- An open database has a tablespace named users that is made up of datafiles all located on the same disk.
- The datafiles of the users tablespace are to be relocated to different and separate disk drives.
- You are currently connected with administrator privileges to the open database. You have a current backup of the database.

Complete the following steps:

1. If you do not know the specific file names or sizes, you can obtain this information by issuing the following query of the data dictionary view DBA_DATAFILES:

   ```sql
   SQL> SELECT FILE_NAME, BYTES FROM DBA_DATAFILES
   2> WHERE TABLESPACE_NAME = 'USERS';
   ```

<table>
<thead>
<tr>
<th>FILE_NAME</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>/u02/oracle/rbdb1/users01.dbf</td>
<td>102400000</td>
</tr>
<tr>
<td>/u02/oracle/rbdb1/users02.dbf</td>
<td>102400000</td>
</tr>
</tbody>
</table>

2. Take the tablespace containing the datafiles offline:

   ```sql
   ALTER TABLESPACE users OFFLINE NORMAL;
   ```

3. Copy the datafiles to their new locations and rename them using the operating system.

   **Note:** You can temporarily exit SQL*Plus to execute an operating system command to copy a file by using the SQL*Plus HOST command.

4. Rename the datafiles within the database.

   The datafile pointers for the files that make up the users tablespace, recorded in the control file of the associated database, must now be changed from the old names to the new names.

   Use the ALTER TABLESPACE...RENAME DATAFILE statement.
ALTER TABLESPACE users RENAME DATAFILE '/u02/oracle/rbdb1/users01.dbf',
'/u02/oracle/rbdb1/users02.dbf' TO '/u03/oracle/rbdb1/users01.dbf',
'/u04/oracle/rbdb1/users02.dbf';

5. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

Procedure for Renaming and Relocating Datafiles in Multiple Tablespaces
You can rename and relocate datafiles in one or more tablespaces using the ALTER DATABASE RENAME FILE statement. This method is the only choice if you want to rename or relocate datafiles of several tablespaces in one operation. You must have the ALTER DATABASE system privilege.

Note: To rename or relocate datafiles of the SYSTEM tablespace, the default temporary tablespace, or the active undo tablespace you must use this ALTER DATABASE method because you cannot take these tablespaces offline.

To rename datafiles in multiple tablespaces, follow these steps.
1. Ensure that the database is mounted but closed.

Note: Optionally, the database does not have to be closed, but the datafiles (or tempfiles) must be offline.

2. Copy the datafiles to be renamed to their new locations and new names, using the operating system.
3. Use ALTER DATABASE to rename the file pointers in the database control file.
For example, the following statement renames the datafiles/u02/oracle/rbdb1/sort01.dbf and /u02/oracle/rbdb1/user3.dbf to /u02/oracle/rbdb1/temp01.dbf and /u02/oracle/rbdb1/users03.dbf, respectively:

ALTER DATABASE RENAME FILE '/u02/oracle/rbdb1/sort01.dbf',
'/u02/oracle/rbdb1/user3.dbf' TO '/u02/oracle/rbdb1/temp01.dbf',
'/u02/oracle/rbdb1/users03.dbf';

Always provide complete filenames (including their paths) to properly identify the old and new datafiles. In particular, specify the old datafile names exactly as they appear in the DBA_DATA_FILES view.
4. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

9.11. Dropping Datafiles
You use the DROP DATAFILE and DROP TEMPFILE clauses of the ALTER TABLESPACE command to drop a single datafile or tempfile. The datafile must be empty. (A datafile is considered to be empty when no extents remain allocated from it.) When you drop a datafile or tempfile, references to the datafile or tempfile are removed from the data dictionary and control files, and the physical file is deleted from the file system or Automatic Storage Management (ASM) disk group. The following example drops the datafile identified by the alias example_df3.f in the ASM disk group DGROUP1. The datafile belongs to the example tablespace.

ALTER TABLESPACE example DROP DATAFILE '+DGROUP1/example_df3.f';

The next example drops the tempfile ltemp02.dbf, which belongs to the ltemp tablespace.

ALTER TABLESPACE ltemp DROP TEMPFILE '/u02/oracle/data/ltemp02.dbf';

This is equivalent to the following statement:

ALTER DATABASE TEMPFILE '/u02/oracle/data/ltemp02.dbf' DROP INCLUDING DATAFILES;
9.11.1. Restrictions for Dropping Datafiles

The following are restrictions for dropping datafiles and tempfiles:

- The database must be open.
- If a datafile is not empty, it cannot be dropped.
- If you must remove a datafile that is not empty and that cannot be made empty by dropping schema objects, you must drop the tablespace that contains the datafile.
- You cannot drop the first or only datafile in a tablespace. This means that DROP DATAFILE cannot be used with a bigfile tablespace.
- You cannot drop datafiles in a read-only tablespace.
- You cannot drop datafiles in the SYSTEM tablespace.
- If a datafile in a locally managed tablespace is offline, it cannot be dropped.

9.12. Verifying Data Blocks in Datafiles

If you want to configure the database to use checksums to verify data blocks, set the initialization parameter DB_BLOCK_CHECKSUM to TYPICAL (the default). This causes the DBWn process and the direct loader to calculate a checksum for each block and to store the checksum in the block header when writing the block to disk. The checksum is verified when the block is read, but only if DB_BLOCK_CHECKSUM is TRUE and the last write of the block stored a checksum. If corruption is detected, the database returns message ORA-01578 and writes information about the corruption to the alert log. The value of the DB_BLOCK_CHECKSUM parameter can be changed dynamically using the ALTER SYSTEM statement. Regardless of the setting of this parameter, checksums are always used to verify data blocks in the SYSTEM tablespace.
10. Undo Management

Beginning with Release 11g, for a default installation, Oracle Database automatically manages undo. There is typically no need for DBA intervention. However, if your installation uses Oracle Flashback operations, you may need to perform some undo management tasks to ensure the success of these operations.

10.1. What Is Undo?

Oracle Database creates and manages information that is used to roll back, or undo, changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. These records are collectively referred to as undo. Undo records are used to:

- Roll back transactions when a ROLLBACK statement is issued
- Recover the database
- Provide read consistency
- Analyze data as of an earlier point in time by using Oracle Flashback Query
- Recover from logical corruptions using Oracle Flashback features

When a ROLLBACK statement is issued, undo records are used to undo changes that were made to the database by the uncommitted transaction. During database recovery, undo records are used to undo any uncommitted changes applied from the redo log to the datafiles. Undo records provide read consistency by maintaining the before image of the data for users who are accessing the data at the same time that another user is changing it.

10.2. Introduction to Automatic Undo Management

10.2.1. Overview of Automatic Undo Management

Oracle provides a fully automated mechanism, referred to as automatic undo management, for managing undo information and space. With automatic undo management, the database manages undo segments in an undo tablespace. Beginning with Release 11g, automatic undo management is the default mode for a newly installed database. An auto-extending undo tablespace named UNDOTBS1 is automatically created when you create the database with Database Configuration Assistant (DBCA).

An undo tablespace can also be created explicitly. When the instance starts, the database automatically selects the first available undo tablespace. If no undo tablespace is available, the instance starts without an undo tablespace and stores undo records in the SYSTEM tablespace. This is not recommended, and an alert message is written to the alert log file to warn that the system is running without an undo tablespace. If the database contains multiple undo tablespaces, you can optionally specify at startup that you want to use a specific undo tablespace. This is done by setting the UNDO_TABLESPACE initialization parameter, as shown in this example:

```
UNDO_TABLESPACE = undotbs_01
```

If the tablespace specified in the initialization parameter does not exist, the STARTUP command fails. The UNDO_TABLESPACE parameter can be used to assign a specific undo tablespace to an instance in an Oracle Real Application Clusters environment. The database can also run in manual undo management mode. In this mode, undo space is managed through rollback segments, and no undo tablespace is used.

**Note:** Space management for rollback segments is complex. Oracle strongly recommends leaving the database in automatic undo management mode.

The following is a summary of the initialization parameters for undo management:

<table>
<thead>
<tr>
<th>Initialization Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

www.wilshiresoft.com           Wilshire Software Technologies
info@wilshiresoft.com          Ph: 2761-2214 / 5577-2214
<table>
<thead>
<tr>
<th>UNDO_MANAGEMENT</th>
<th>If AUTO or null, enables automatic undo management. If MANUAL, sets manual undo management mode. The default is AUTO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDO_TABLESPACE</td>
<td>Optional and valid only in automatic undo management mode. Specifies the name of an undo tablespace. Use only when the database has multiple undo tablespaces and you want to direct the database instance to use a particular undo tablespace.</td>
</tr>
</tbody>
</table>

When automatic undo management is enabled, if the initialization parameter file contains parameters relating to manual undo management, they are ignored.

**Note:** Earlier releases of Oracle Database default to manual undo management mode. To change to automatic undo management, you must first create an undo tablespace and then change the UNDO_MANAGEMENT initialization parameter to AUTO. If your Oracle Database is release 9i or later and you want to change to automatic undo management in Release 11g and later, but defaults to manual undo management mode in earlier releases. You must therefore use caution when upgrading a previous release to Release 11g. Oracle Database Upgrade Guide describes the correct method of migrating to automatic undo management mode, including information on how to size the undo tablespace.

### 10.3. About the Undo Retention Period

After a transaction is committed, undo data is no longer needed for rollback or transaction recovery purposes. However, for consistent read purposes, long-running queries may require this old undo information for producing older images of data blocks. Furthermore, the success of several Oracle Flashback features can also depend upon the availability of older undo information. For these reasons, it is desirable to retain the old undo information for as long as possible.

When automatic undo management is enabled, there is always a current undo retention period, which is the minimum amount of time that Oracle Database attempts to retain old undo information before overwriting it. Old (committed) undo information that is older than the current undo retention period is said to be expired and its space is available to be overwritten by new transactions. Old undo information with an age that is less than the current undo retention period is said to be unexpired and is retained for consistent read and Oracle Flashback operations. Oracle Database automatically tunes the undo retention period based on undo tablespace size and system activity. You can optionally specify a minimum undo retention period (in seconds) by setting the UNDO_RETENTION initialization parameter. The exact impact this parameter on undo retention is a follows:

The UNDO_RETENTION parameter is ignored for a fixed size undo tablespace. The database always tunes the undo retention period for the best possible retention, based on system activity and undo tablespace size.

For an undo tablespace with the AUTOEXTEND option enabled, the database attempts to honor the minimum retention period specified by UNDO_RETENTION. When space is low, instead of overwriting unexpired undo information, the tablespace auto-extends. If the MAXSIZE clause is specified for an auto-extending undo tablespace, when the maximum size is reached, the database may begin to overwrite unexpired undo information. The UNDOTBS1 tablespace that is automatically created by DBCA is auto-extending.

### 10.3.1 Automatic Tuning of Undo Retention

Oracle Database automatically tunes the undo retention period based on how the undo tablespace is configured.

- If the undo tablespace is configured with the AUTOEXTEND option, the database dynamically tunes the undo retention period to be somewhat longer than the longest-running active query on the system. However, this retention period may be insufficient to accommodate Oracle Flashback operations. Oracle Flashback operations resulting in snapshot too old errors are the indicator that you must intervene to ensure that sufficient undo data is retained to support these operations. To better accommodate Oracle
Flashback features, you can either set the UNDO_RETENTION parameter to a value equal to the longest expected Oracle Flashback operation, or you can change the undo tablespace to fixed size.

- If the undo tablespace is fixed size, the database dynamically tunes the undo retention period for the best possible retention for that tablespace size and the current system load. This best possible retention time is typically significantly greater than the duration of the longest-running active query.
- If you decide to change the undo tablespace to fixed-size, you must choose a tablespace size that is sufficiently large. If you choose an undo tablespace size that is too small, the following two errors could occur:
  - DML could fail because there is not enough space to accommodate undo for new transactions.
  - Long-running queries could fail with a snapshot too old error, which means that there was insufficient undo data for read consistency.

Note: Automatic tuning of undo retention is not supported for LOBs. This is because undo information for LOBs is stored in the segment itself and not in the undo tablespace. For LOBs, the database attempts to honor the minimum undo retention period specified by UNDO_RETENTION. However, if space becomes low, unexpired LOB undo information may be overwritten.

**10.3.2. Retention Guarantee**

To guarantee the success of long-running queries or Oracle Flashback operations, you can enable retention guarantee. If retention guarantee is enabled, the specified minimum undo retention is guaranteed; the database never overwrites unexpired undo data even if it means that transactions fail due to lack of space in the undo tablespace. If retention guarantee is not enabled, the database can overwrite unexpired undo when space is low, thus lowering the undo retention for the system. This option is disabled by default.

**Warning:** Enabling retention guarantee can cause multiple DML operations to fail.

You enable retention guarantee by specifying the RETENTION GUARANTEE clause for the undo tablespace when you create it with either the CREATE DATABASE or CREATE UNDO TABLESPACE statement. Or, you can later specify this clause in an ALTER TABLESPACE statement. You disable retention guarantee with the RETENTION NOGUARANTEE clause. You can use the DBA_TABLESPACES view to determine the retention guarantee setting for the undo tablespace. A column named RETENTION contains a value of GUARANTEE, NOGUARantee, or NOT APPLY, where NOT APPLY is used for tablespaces other than the undo tablespace.

**10.3.3. Undo Retention Tuning and Alert Thresholds**

For a fixed-size undo tablespace, the database calculates the best possible retention based on database statistics and on the size of the undo tablespace. For optimal undo management, rather than tuning based on 100% of the tablespace size, the database tunes the undo retention period based on 85% of the tablespace size or on the warning alert threshold percentage for space used, whichever is lower. (The warning alert threshold defaults to 85%, but can be changed.) Therefore, if you set the warning alert threshold of the undo tablespace below 85%, this may reduce the tuned size of the undo retention period.

**10.3.4. Tracking the Tuned Undo Retention Period**

You can determine the current retention period by querying the TUNED_UNDORETENTION column of the V$UNDOSTAT view. This view contains one row for each 10-minute statistics collection interval over the last 4 days. (Beyond 4 days, the data is available in the DBA_HIST_UNDOSTAT view.) TUNED_UNDORETENTION is given in seconds. Select to_char (begin_time, 'DD-MON-RR HH24:MI') begin_time, to_char(end_time, 'DD-MON-RR HH24:MI') end_time, tuned_undoretention from v$undostat order by end_time;

```sql
BEGIN_TIME      END_TIME        TUNED_UNDORETENTION
--------------- --------------- -------------------
04-FEB-05 00:01 04-FEB-05 00:11               12100
...
```
10.3.5. Setting the Minimum Undo Retention Period

You specify the minimum undo retention period (in seconds) by setting the UNDO_RETENTION initialization parameter. The current undo retention period may be automatically tuned to be greater than UNDO_RETENTION, or, unless retention guarantee is enabled, less than UNDO_RETENTION if space in the undo tablespace is low. To set the minimum undo retention period:

Do one of the following:

Set UNDO_RETENTION in the initialization parameter file.

```
UNDO_RETENTION = 1800
```

Change UNDO_RETENTION at any time using the ALTER SYSTEM statement:

```
ALTER SYSTEM SET UNDO_RETENTION = 2400;
```

The effect of an UNDO_RETENTION parameter change is immediate, but it can only be honored if the current undo tablespace has enough space.

10.4. Sizing a Fixed-Size Undo Tablespace

If you have decided on a fixed-size undo tablespace, the Undo Advisor can help you estimate needed capacity. You can access the Undo Advisor through Enterprise Manager or through the DBMS_ADVISOR PL/SQL package. Enterprise Manager is the preferred method of accessing the advisor. The Undo Advisor relies for its analysis on data collected in the Automatic Workload Repository (AWR). It is therefore important that the AWR have adequate workload statistics available so that the Undo Advisor can make accurate recommendations. For newly created databases, adequate statistics may not be available immediately. In such cases, an auto-extending undo tablespace can be used. An adjustment to the collection interval and retention period for AWR statistics can affect the precision and the type of recommendations that the advisor produces. To use the Undo Advisor, you first estimate these two values:

- The length of your expected longest running query After the database has completed a workload cycle, you can view the Longest Running Query field on the System Activity subpage of the Automatic Undo Management page.
- The longest interval that you will require for Oracle Flashback operations

For example, if you expect to run Oracle Flashback queries for up to 48 hours in the past, your Oracle Flashback requirement is 48 hours. You then take the maximum of these two values and use that value as input to the Undo Advisor. Running the Undo Advisor does not alter the size of the undo tablespace. The advisor just returns a recommendation. You must use ALTER DATABASE statements to change the tablespace datafiles to fixed sizes.

```
ALTER DATABASE DATAFILE '/oracle/dbs/undotbs.dbf' RESIZE 300M;
ALTER DATABASE DATAFILE '/oracle/dbs/undotbs.dbf' AUTOEXTEND OFF;
```

Note: If you want to make the undo tablespace fixed-size, Oracle suggests that you first allow enough time after database creation to run a full workload, thus allowing the undo tablespace to grow to its minimum required size to handle the workload. Then, you can use the Undo Advisor to determine, if desired, how much larger to set the size of the undo tablespace to allow for long-running queries and Oracle Flashback operations.
10.5. Creating an Undo Tablespace

Although Database Configuration Assistant (DBCA) automatically creates an undo tablespace for new installations of Oracle Database Release 11g, there may be occasions when you want to manually create an undo tablespace. There are two methods of creating an undo tablespace. The first method creates the undo tablespace when the CREATE DATABASE statement is issued. This occurs when you are creating a new database, and the instance is started in automatic undo management mode (UNDO_MANAGEMENT = AUTO). The second method is used with an existing database. It uses the CREATE UNDO TABLESPACE statement. You cannot create database objects in an undo tablespace. It is reserved for system-managed undo data.

10.5.1. Using CREATE DATABASE to Create an Undo Tablespace

You can create a specific undo tablespace using the UNDO TABLESPACE clause of the CREATE DATABASE statement. The following statement illustrates using the UNDO TABLESPACE clause in a CREATE DATABASE statement. The undo tablespace is named undotbs_01 and one datafile, /u01/oracle/rbdb1/undo0101.dbf, is allocated for it.

```sql
CREATE DATABASE rbdb1
    CONTROLFILE REUSE
    .
    .
    UNDO TABLESPACE undotbs_01 DATAFILE '/u01/oracle/rbdb1/undo0101.dbf';
```

If the undo tablespace cannot be created successfully during CREATE DATABASE, the entire CREATE DATABASE operation fails. You must clean up the database files, correct the error and retry the CREATE DATABASE operation. The CREATE DATABASE statement also lets you create a single-file undo tablespace at database creation.

10.5.2. Using the CREATE UNDO TABLESPACE Statement

The CREATE UNDO TABLESPACE statement is the same as the CREATE TABLESPACE statement, but the UNDO keyword is specified. The database determines most of the attributes of the undo tablespace, but you can specify the DATAFILE clause. This example creates the undotbs_02 undo tablespace with the AUTOEXTEND option:

```sql
CREATE UNDO TABLESPACE undotbs_02 DATAFILE '/u01/oracle/rbdb1/undo0201.dbf' SIZE 2M
    REUSE AUTOEXTEND ON;
```

You can create more than one undo tablespace, but only one of them can be active at any one time.

10.6. Altering an Undo Tablespace

Undo tablespaces are altered using the ALTER TABLESPACE statement. However, since most aspects of undo tablespaces are system managed, you need only be concerned with the following actions:

- Adding a datafile
- Renaming a datafile
- Bringing a datafile online or taking it offline
- Beginning or ending an open backup on a datafile
- Enabling and disabling undo retention guarantee

These are also the only attributes you are permitted to alter.

If an undo tablespace runs out of space, or you want to prevent it from doing so, you can add more files to it or resize existing datafiles. The following example adds another datafile to undo tablespace undotbs_01:

```sql
ALTER TABLESPACE undotbs_01 ADD DATAFILE '/u01/oracle/rbdb1/undo0102.dbf' AUTOEXTEND
    ON NEXT 1M MAXSIZE UNLIMITED;
```
You can use the ALTER DATABASE...DATAFILE statement to resize or extend a datafile.

10.7. Dropping an Undo Tablespace

Use the DROP TABLESPACE statement to drop an undo tablespace. The following example drops the undo tablespace undotbs_01:

```
DROP TABLESPACE undotbs_01;
```

An undo tablespace can only be dropped if it is not currently used by any instance. If the undo tablespace contains any outstanding transactions (for example, a transaction died but has not yet been recovered), the DROP TABLESPACE statement fails. However, since DROP TABLESPACE drops an undo tablespace even if it contains unexpired undo information (within retention period), you must be careful not to drop an undo tablespace if undo information is needed by some existing queries. DROP TABLESPACE for undo tablespaces behaves like DROP TABLESPACE...INCLUDING CONTENTS. All contents of the undo tablespace are removed.

10.8. Switching Undo Tablespaces

You can switch from using one undo tablespace to another. Because the UNDO_TABLESPACE initialization parameter is a dynamic parameter, the ALTER SYSTEM SET statement can be used to assign a new undo tablespace. The following statement switches to a new undo tablespace:

```
ALTER SYSTEM SET UNDO_TABLESPACE = undotbs_02;
```

Assuming undotbs_01 is the current undo tablespace, after this command successfully executes, the instance uses undotbs_02 in place of undotbs_01 as its undo tablespace. If any of the following conditions exist for the tablespace being switched to, an error is reported and no switching occurs:

- The tablespace does not exist
- The tablespace is not an undo tablespace
- The tablespace is already being used by another instance (in a RAC environment only)

The database is online while the switch operation is performed, and user transactions can be executed while this command is being executed. When the switch operation completes successfully, all transactions started after the switch operation began are assigned to transaction tables in the new undo tablespace. The switch operation does not wait for transactions in the old undo tablespace to commit. If there are any pending transactions in the old undo tablespace, the old undo tablespace enters into a PENDING OFFLINE mode (status). In this mode, existing transactions can continue to execute, but undo records for new user transactions cannot be stored in this undo tablespace.

An undo tablespace can exist in this PENDING OFFLINE mode, even after the switch operation completes successfully. A PENDING OFFLINE undo tablespace cannot be used by another instance, nor can it be dropped. Eventually, after all active transactions have committed, the undo tablespace automatically goes from the PENDING OFFLINE mode to the OFFLINE mode. From then on, the undo tablespace is available for other instances (in an Oracle Real Application Cluster environment). If the parameter value for UNDO_TABLESPACE is set to " (two single quotes), then the current undo tablespace is switched out and the next available undo tablespace is switched in. Use this statement with care because there may be no undo tablespace available. The following example unassigns the current undo tablespace:

```
ALTER SYSTEM SET UNDO_TABLESPACE = '';
```

10.9. Establishing User Quotas for Undo Space

The Oracle Database Resource Manager can be used to establish user quotas for undo space. The Database Resource Manager directive UNDO_POOL allows DBAs to limit the amount of undo space consumed by a group of
users (resource consumer group). You can specify an undo pool for each consumer group. An undo pool controls the amount of total undo that can be generated by a consumer group. When the total undo generated by a consumer group exceeds its undo limit, the current UPDATE transaction generating the undo is terminated. No other members of the consumer group can perform further updates until undo space is freed from the pool. When no UNDO_POOL directive is explicitly defined, users are allowed unlimited undo space.

10.10. Managing Space Threshold Alerts for the Undo Tablespace

Oracle Database also provides proactive help in managing tablespace disk space use by alerting you when tablespaces run low on available space. In addition to the proactive undo space alerts, Oracle Database also provides alerts if your system has long-running queries that cause SNAPSHOT TOO OLD errors. To prevent excessive alerts, the long query alert is issued at most once every 24 hours.

10.11. Migrating to Automatic Undo Management

If you are currently using rollback segments to manage undo space, Oracle strongly recommends that you migrate your database to automatic undo management.

10.12. Undo Space Data Dictionary Views

This section lists views that are useful for viewing information about undo space in the automatic undo management mode and provides some examples. In addition to views listed here, you can obtain information from the views available for viewing tablespace and datafile information. The following dynamic performance views are useful for obtaining space information about the undo tablespace:

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V$UNDOSTAT</td>
<td>Contains statistics for monitoring and tuning undo space. Use this view to help estimate the amount of undo space required for the current workload. The database also uses this information to help tune undo usage in the system. This view is meaningful only in automatic undo management mode.</td>
</tr>
<tr>
<td>V$ROLLSTAT</td>
<td>For automatic undo management mode, information reflects behavior of the undo segments in the undo tablespace</td>
</tr>
<tr>
<td>V$TRANSACTION</td>
<td>Contains undo segment information</td>
</tr>
<tr>
<td>DBA_UNDO_EXTENTS</td>
<td>Shows the status and size of each extent in the undo tablespace.</td>
</tr>
<tr>
<td>DBA_HIST_UNDOSTAT</td>
<td>Contains statistical snapshots of V$UNDOSTAT information.</td>
</tr>
</tbody>
</table>

The V$UNDOSTAT view is useful for monitoring the effects of transaction execution on undo space in the current instance. Statistics are available for undo space consumption, transaction concurrency, the tuning of undo retention, and the length and SQL ID of long-running queries in the instance. Each row in the view contains statistics collected in the instance for a ten-minute interval. The rows are in descending order by the BEGIN_TIME column value. Each row belongs to the time interval marked by (BEGIN_TIME, END_TIME). Each column represents the data collected for the particular statistic in that time interval. The first row of the view contains statistics for the (partial) current time period. The view contains a total of 576 rows, spanning a 4 day cycle. The following example shows the results of a query on the V$UNDOSTAT view.

```
SELECT TO_CHAR(BEGIN_TIME, 'MM/DD/YYYY HH24:MI:SS') BEGIN_TIME,
       TO_CHAR(END_TIME, 'MM/DD/YYYY HH24:MI:SS') END_TIME,
       UNDOTSN, UNDOBLKS, TXNCOUNT, MAXCONCURRENCY AS "MAXCON"
FROM v$UNDOSTAT WHERE rownum <= 144;
```

<table>
<thead>
<tr>
<th>BEGIN_TIME</th>
<th>END_TIME</th>
<th>UNDOTSN</th>
<th>UNDOBLKS</th>
<th>TXNCOUNT</th>
<th>MAXCON</th>
</tr>
</thead>
</table>
The preceding example shows how undo space is consumed in the system for the previous 24 hours from the time 14:35:12 on 10/27/2004.

```
<table>
<thead>
<tr>
<th>Start Date</th>
<th>Start Time</th>
<th>End Date</th>
<th>End Time</th>
<th>Rows</th>
<th>File Size</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

144 rows selected.
11. Oracle-Managed Files (OMF)

11.1. What Are Oracle-Managed Files?

Using Oracle-managed files simplifies the administration of an Oracle Database. Oracle-managed files eliminate the need for you, the DBA, to directly manage the operating system files that make up an Oracle Database. With Oracle-managed files, you specify file system directories in which the database automatically creates, names, and manages files at the database object level. For example, you need only specify that you want to create a tablespace; you do not need to specify the name and path of the tablespace's datafile with the DATAFILE clause. This feature works well with a logical volume manager (LVM). The database internally uses standard file system interfaces to create and delete files as needed for the following database structures:

- Tablespaces
- Redo log files
- Control files
- Archived logs
- Block change tracking files
- Flashback logs
- RMAN backups

Through initialization parameters, you specify the file system directory to be used for a particular type of file. The database then ensures that a unique file, an Oracle-managed file, is created and deleted when no longer needed. This feature does not affect the creation or naming of administrative files such as trace files, audit files, alert logs, and core files.

11.2. Who Can Use Oracle-Managed Files?

Oracle-managed files are most useful for the following types of databases:

Databases that are supported by the following:

- A logical volume manager that supports striping/RAID and dynamically extensible logical volumes
- A file system that provides large, extensible files
- Low end or test databases

The Oracle-managed files feature is not intended to ease administration of systems that use raw disks. This feature provides better integration with operating system functionality for disk space allocation. Since there is no operating system support for allocation of raw disks (it is done manually), this feature cannot help. On the other hand, because Oracle-managed files require that you use the operating system file system (unlike raw disks), you lose control over how files are laid out on the disks and thus, you lose some I/O tuning ability.

11.3. What Is a Logical Volume Manager?

A logical volume manager (LVM) is a software package available with most operating systems. Sometimes it is called a logical disk manager (LDM). It allows pieces of multiple physical disks to be combined into a single contiguous address space that appears as one disk to higher layers of software. An LVM can make the logical volume have better capacity, performance, reliability, and availability characteristics than any of the underlying physical disks. It uses techniques such as mirroring, striping, concatenation, and RAID 5 to implement these characteristics. Some LVMs allow the characteristics of a logical volume to be changed after it is created, even while it is in use. The volume may be resized or mirrored, or it may be relocated to different physical disks.
11.4. What Is a File System?

A file system is a data structure built inside a contiguous disk address space. A file manager (FM) is a software package that manipulates file systems, but it is sometimes called the file system. All operating systems have file managers. The primary task of a file manager is to allocate and deallocate disk space into files within a file system. A file system allows the disk space to be allocated to a large number of files. Each file is made to appear as a contiguous address space to applications such as Oracle Database. The files may not actually be contiguous within the disk space of the file system. Files can be created, read, written, resized, and deleted. Each file has a name associated with it that is used to refer to the file. A file system is commonly built on top of a logical volume constructed by an LVM. Thus all the files in a particular file system have the same performance, reliability, and availability characteristics inherited from the underlying logical volume. A file system is a single pool of storage that is shared by all the files in the file system. If a file system is out of space, then none of the files in that file system can grow. Space available in one file system does not affect space in another file system. However some LVM/FM combinations allow space to be added or removed from a file system. An operating system can support multiple file systems. Multiple file systems are constructed to give different storage characteristics to different files as well as to divide the available disk space into pools that do not affect each other.

11.5. Benefits of Using Oracle-Managed Files

Consider the following benefits of using Oracle-managed files:

- They make the administration of the database easier. There is no need to invent filenames and define specific storage requirements. A consistent set of rules is used to name all relevant files. The file system defines the characteristics of the storage and the pool where it is allocated.

- They reduce corruption caused by administrators specifying the wrong file. Each Oracle-managed file and filename is unique. Using the same file in two different databases is a common mistake that can cause very large down times and loss of committed transactions. Using two different names that refer to the same file is another mistake that causes major corruptions.

- They reduce wasted disk space consumed by obsolete files. Oracle Database automatically removes old Oracle-managed files when they are no longer needed. Much disk space is wasted in large systems simply because no one is sure if a particular file is still required. This also simplifies the administrative task of removing files that are no longer required on disk and prevents the mistake of deleting the wrong file.

- They simplify creation of test and development databases. You can minimize the time spent making decisions regarding file structure and naming, and you have fewer file management tasks. You can focus better on meeting the actual requirements of your test or development database.

- Oracle-managed files make development of portable third-party tools easier. Oracle-managed files eliminate the need to put operating system specific file names in SQL scripts.

11.6. Oracle-Managed Files and Existing Functionality

Using Oracle-managed files does not eliminate any existing functionality. Existing databases are able to operate as they always have. New files can be created as managed files while old ones are administered in the old way. Thus, a database can have a mixture of Oracle-managed and unmanaged files.

11.7. Enabling the Creation and Use of Oracle-Managed Files

The following initialization parameters allow the database server to use the Oracle-managed files feature:
<table>
<thead>
<tr>
<th>Initialization Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_CREATE_FILE_DEST</td>
<td>Defines the location of the default file system directory where the database creates datafiles or tempfiles when no file specification is given in the creation operation. Also used as the default file system directory for redo log and control files if DB_CREATE_ONLINE_LOG_DEST_n is not specified.</td>
</tr>
<tr>
<td>DB_CREATE_ONLINE_LOG_DEST_n</td>
<td>Defines the location of the default file system directory for redo log files and control file creation when no file specification is given in the creation operation. You can use this initialization parameter multiple times, where n specifies a multiplexed copy of the redo log or control file. You can specify up to five multiplexed copies.</td>
</tr>
<tr>
<td>DB_RECOVERY_FILE_DEST</td>
<td>Defines the location of the flash recovery area, which is the default file system directory where the database creates RMAN backups when no format option is used, archived logs when no other local destination is configured, and flashback logs. Also used as the default file system directory for redo log and control files if DB_CREATE_ONLINE_LOG_DEST_n is not specified.</td>
</tr>
</tbody>
</table>

The file system directory specified by either of these parameters must already exist: the database does not create it. The directory must also have permissions to allow the database to create the files in it. The default location is used whenever a location is not explicitly specified for the operation creating the file. The database creates the filename, and a file thus created is an Oracle-managed file. Both of these initialization parameters are dynamic, and can be set using the ALTER SYSTEM or ALTER SESSION statement.

### 11.7.1. Setting the DB_CREATE_FILEDEST Initialization Parameter

Include the DB_CREATE_FILEDEST initialization parameter in your initialization parameter file to identify the default location for the database server to create:

- Datafiles
- Tempfiles
- Redo log files
- Control files
- Block change tracking files

You specify the name of a file system directory that becomes the default location for the creation of the operating system files for these entities. The following example sets `/u01/oradata` as the default directory to use when creating Oracle-managed files:

```
DB_CREATE_FILEDEST = '/u01/oradata'
```

### 11.7.2. Setting the DB_RECOVERY_FILEDEST Parameter

Include the DB_RECOVERY_FILEDEST and DB_RECOVERY_FILEDEST_SIZE parameters in your initialization parameter file to identify the default location in which Oracle Database should create:

- Redo log files
- Control files
- RMAN backups (datafile copies, control file copies, backup pieces, control file autobackups)
- Archived logs
• Flashback logs

You specify the name of file system directory that becomes the default location for creation of the operating system files for these entities. For example:

```
DB_RECOVERY_FILE_DEST = '/u01/oradata'
DB_RECOVERY_FILE_DEST_SIZE = 20G
```

Include the `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameter in your initialization parameter file to identify the default location for the database server to create:

• Redo log files
• Control files

You specify the name of a file system directory that becomes the default location for the creation of the operating system files for these entities. You can specify up to five multiplexed locations. For the creation of redo log files and control files only, this parameter overrides any default location specified in the `DB_CREATE_FILE_DEST` and `DB_RECOVERY_FILE_DEST` initialization parameters. If you do not specify a `DB_CREATE_FILE_DEST` parameter, but you do specify the `DB_CREATE_ONLINE_LOG_DEST_n` parameter, then only redo log files and control files can be created as Oracle-managed files. It is recommended that you specify at least two parameters. For example:

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

This allows multiplexing, which provides greater fault-tolerance for the redo log and control file if one of the destinations fails.

### 11.8. Creating Oracle-Managed Files

If you have met any of the following conditions, then Oracle Database creates Oracle-managed files for you, as appropriate, when no file specification is given in the creation operation:

• You have included any of the `DB_CREATE_FILE_DEST`, `DB_RECOVERY_FILE_DEST`, or `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters in your initialization parameter file.
• You have issued the `ALTER SYSTEM` statement to dynamically set any of `DB_RECOVERY_FILE_DEST`, `DB_CREATE_FILE_DEST`, or `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters.
• You have issued the `ALTER SESSION` statement to dynamically set any of the `DB_CREATE_FILE_DEST`, `DB_RECOVERY_FILE_DEST`, or `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters.

If a statement that creates an Oracle-managed file finds an error or does not complete due to some failure, then any Oracle-managed files created by the statement are automatically deleted as part of the recovery of the error or failure. However, because of the large number of potential errors that can occur with file systems and storage subsystems, there can be situations where you must manually remove the files using operating system commands.

### 11.8.1. How Oracle-Managed Files Are Named

The filenames of Oracle-managed files comply with the Optimal Flexible Architecture (OFA) standard for file naming. The assigned names are intended to meet the following requirements:

• Database files are easily distinguishable from all other files.
• Files of one database type are easily distinguishable from other database types.
• Files are clearly associated with important attributes specific to the file type.

For example, a datafile name may include the tablespace name to allow for easy association of datafile to tablespace, or an archived log name may include the thread, sequence, and creation date. No two Oracle-managed files are given the same name. The name that is used for creation of an Oracle-managed file is constructed from three sources:

• The default creation location
• A file name template that is chosen based on the type of the file. The template also depends on the operating system platform and whether or not automatic storage management is used.
• A unique string created by the Oracle Database server or the operating system.

This ensures that file creation does not damage an existing file and that the file cannot be mistaken for some other file. As a specific example, filenames for Oracle-managed files have the following format on a Solaris file system:

\(<destination\ prefix>/o1\_mf\_%t\_%u\_.dbf\)

Where:
\(<destination\ prefix>\) is \(<destination\ location>/\<db\_unique\ name>/\<datafile>\)

Where:
– \(<destination\ location>\) is the location specified in DB_CREATE_FILE_DEST
– \(<db\_unique\ name>\) is the globally unique name (DB_UNIQUE_NAME initialization parameter) of the target database. If there is no DB_UNIQUE_NAME parameter, then the DB_NAME initialization parameter value is used.
%t is the tablespace name.
%u is an eight-character string that guarantees uniqueness

For example, assume the following parameter settings:

DB_CREATE_FILE_DEST = /u01/oradata
DB_UNIQUE_NAME = PAYROLL

Then an example datafile name would be:

/u01/oradata/PAYROLL/datafile/o1\_mf\_tbsl\_21xh90q\_.dbf

Names for other file types are similar. Names on other platforms are also similar, subject to the constraints of the naming rules of the platform. The examples on the following pages use Oracle-managed file names as they might appear with a Solaris file system as an OMF destination.

Caution: Do not rename an Oracle-managed file. The database identifies an Oracle-managed file based on its name. If you rename the file, the database is no longer able to recognize it as an Oracle-managed file and will not manage the file accordingly.

11.8.2. Creating Oracle-Managed Files at Database Creation

The behavior of the CREATE DATABASE statement for creating database structures when using Oracle-managed files is discussed in this section.

Specifying Control Files at Database Creation

At database creation, the control file is created in the files specified by the CONTROL_FILES initialization parameter. If the CONTROL_FILES parameter is not set and at least one of the initialization parameters required for the creation of Oracle-managed files is set, then an Oracle-managed control file is created in the default control file destinations. In order of precedence, the default destination is defined as follows:

• One or more control files as specified in the DB_CREATE_ONLINE_LOG_DEST_n initialization parameter. The file in the first directory is the primary control file. When
DB_CREATE_ONLINE_LOG_DEST_n is specified, the database does not create a control file in DB_CREATE_FILE_DEST or in DB_RECOVERY_FILE_DEST (the flash recovery area).

- If no value is specified for DB_CREATE_ONLINE_LOG_DEST_n, but values are set for both the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST, then the database creates one control file in each location. The location specified in DB_CREATE_FILE_DEST is the primary control file.
- If a value is specified only for DB_CREATE_FILE_DEST, then the database creates one control file in that location.
- If a value is specified only for DB_RECOVERY_FILE_DEST, then the database creates one control file in that location.

If the CONTROL_FILES parameter is not set and none of these initialization parameters are set, then the Oracle Database default behavior is operating system dependent. At least one copy of a control file is created in an operating system dependent default location. Any copies of control files created in this fashion are not Oracle-managed files, and you must add a CONTROL_FILES initialization parameter to any initialization parameter file. If the database creates an Oracle-managed control file, and if there is a server parameter file, then the database creates a CONTROL_FILES initialization parameter entry in the server parameter file. If there is no server parameter file, then you must manually include a CONTROL_FILES initialization parameter entry in the text initialization parameter file.

Specifying Redo Log Files at Database Creation

The LOGFILE clause is not required in the CREATE DATABASE statement, and omitting it provides a simple means of creating Oracle-managed redo log files. If the LOGFILE clause is omitted, then redo log files are created in the default redo log file destinations. In order of precedence, the default destination is defined as follows:

- If either the DB_CREATE_ONLINE_LOG_DEST_n is set, then the database creates a log file member in each directory specified, up to the value of the MAXLOGMEMBERS initialization parameter.
- If the DB_CREATE_ONLINE_LOG_DEST_n parameter is not set, but both the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST initialization parameters are set, then the database creates one Oracle-managed log file member in each of those locations. The log file in the DB_CREATE_FILE_DEST destination is the first member.
- If only the DB_CREATE_FILE_DEST initialization parameter is specified, then the database creates a log file member in that location.
- If only the DB_RECOVERY_FILE_DEST initialization parameter is specified, then the database creates a log file member in that location.

The default size of an Oracle-managed redo log file is 100 MB. Optionally, you can create Oracle-managed redo log files, and override default attributes, by including the LOGFILE clause but omitting a filename. Redo log files are created the same way, except for the following: If no filename is provided in the LOGFILE clause of CREATE DATABASE, and none of the initialization parameters required for creating Oracle-managed files are provided, then the CREATE DATABASE statement fails.

Specifying the SYSTEM and SYSAUX Tablespace Datafiles at Database Creation

The DATAFILE or SYSAUX_DATAFILE clause is not required in the CREATE DATABASE statement, and omitting it provides a simple means of creating Oracle-managed datafiles for the SYSTEM and SYSAUX tablespaces. If the DATAFILE clause is omitted, then one of the following actions occurs:

- If DB_CREATE_FILE_DEST is set, then one Oracle-managed datafile for the SYSTEM tablespace and another for the SYSAUX tablespace are created in the DB_CREATE_FILE_DEST directory.
- If DB_CREATE_FILE_DEST is not set, then the database creates one SYSTEM, and one SYSAUX, tablespace datafile whose name and size are operating system dependent. Any SYSTEM or SYSAUX tablespace datafile created in this manner is not an Oracle-managed file.
The default size for an Oracle-managed datafile is 100 MB and the file is autoextensible. When autoextension is required, the database extends the datafile by its existing size or 100 MB, whichever is smaller. You can also explicitly specify the autoextensible unit using the NEXT parameter of the STORAGE clause when you specify the datafile (in a CREATE or ALTER TABLESPACE operation). Optionally, you can create an Oracle-managed datafile for the SYSTEM or SYSAUX tablespace and override default attributes. This is done by including the DATAFILE clause, omitting a filename, but specifying overriding attributes. When a filename is not supplied and the DB_CREATE_FILE_DEST parameter is set, an Oracle-managed datafile for the SYSTEM or SYSAUX tablespace is created in the DB_CREATE_FILE_DEST directory with the specified attributes being overridden. However, if a filename is not supplied and the DB_CREATE_FILE_DEST parameter is not set, then the CREATE DATABASE statement fails. When overriding the default attributes of an Oracle-managed file, if a SIZE value is specified but no AUTOEXTEND clause is specified, then the datafile is not autoextensible.

Specifying the Undo Tablespace Datafile at Database Creation

The DATAFILE subclause of the UNDO TABLESPACE clause is optional and a filename is not required in the file specification. If a filename is not supplied and the DB_CREATE_FILE_DEST parameter is set, then an Oracle-managed datafile is created in the DB_CREATE_FILE_DEST directory. If DB_CREATE_FILE_DEST is not set, then the statement fails with a syntax error. The UNDO TABLESPACE clause itself is optional in the CREATE DATABASE statement. If it is not supplied, and automatic undo management mode is enabled, then a default undo tablespace named SYS_UNDOTBS is created and a 10 MB datafile that is autoextensible is allocated as follows:

- If DB_CREATE_FILE_DEST is set, and then an Oracle-managed datafile is created in the indicated directory.
- If DB_CREATE_FILE_DEST is not set, and then the datafile location is operating system specific.

Specifying the Default Temporary Tablespace Tempfile at Database Creation

The TEMPFILE subclause is optional for the DEFAULT TEMPORARY TABLESPACE clause and a filename is not required in the file specification. If a filename is not supplied and the DB_CREATE_FILE_DEST parameter set, then an Oracle-managed tempfile is created in the DB_CREATE_FILE_DEST directory. If DB_CREATE_FILE_DEST is not set, then the CREATE DATABASE statement fails with a syntax error. The DEFAULT TEMPORARY TABLESPACE clause itself is optional. If it is not specified, then no default temporary tablespace is created. The default size for an Oracle-managed tempfile is 100 MB and the file is autoextensible with an unlimited maximum size.

CREATE DATABASE Statement Using Oracle-Managed Files: Examples

This section contains examples of the CREATE DATABASE statement when using the Oracle-managed files feature.

Example 1: This example creates a database with the following Oracle-managed files:

- A SYSTEM tablespace datafile in directory /u01/oradata that is 100 MB and autoextensible up to an unlimited size.
- A SYSAUX tablespace datafile in directory /u01/oradata that is 100 MB and autoextensible up to an unlimited size. The tablespace is locally managed with automatic segment-space management.
- Two online log groups with two members of 100 MB each, one each in /u02/oradata and /u03/oradata.
- If automatic undo management mode is enabled, then an undo tempfile datafile in directory /u01/oradata that is 10 MB and autoextensible up to an unlimited size. An undo tempfile named SYS_UNDOTBS is created.
- If no CONTROL_FILES initialization parameter is specified, then two control files, one each in /u02/oradata and /u03/oradata. The control file in /u02/oradata is the primary control file.

The following parameter settings relating to Oracle-managed files, are included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'

The following statement is issued at the SQL prompt:

SQL> CREATE DATABASE sample;

Example: this example creates a database with the following Oracle-managed files:

- A 100 MB SYSTEM tablespace datafile in directory /u01/oradata that is autoextensible up to an unlimited size.
- A SYSAUX tablespace datafile in directory /u01/oradata that is 100 MB and autoextensible up to an unlimited size. The tablespace is locally managed with automatic segment-space management.
- Two redo log files of 100 MB each in directory /u01/oradata. They are not multiplexed.
- An undo tablespace datafile in directory /u01/oradata that is 10 MB and autoextensible up to an unlimited size. An undo tablespace named SYS_UNDOTBS is created.
- A control file in /u01/oradata.

In this example, it is assumed that:

- No DB_CREATE_ONLINE_LOG_DEST_n initialization parameters are specified in the initialization parameter file.
- No CONTROL_FILES initialization parameter was specified in the initialization parameter file.
- Automatic undo management mode is enabled.

The following statements are issued at the SQL prompt:

SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE DATABASE sample2;

This database configuration is not recommended for a production database. The example illustrates how a very low-end database or simple test database can easily be created. To better protect this database from failures, at least one more control file should be created and the redo log should be multiplexed.

Example: In this example, the file size for the Oracle-managed files for the default temporary tablespace and undo tablespace are specified. A database with the following Oracle-managed files is created:

- A 400 MB SYSTEM tablespace datafile in directory /u01/oradata. Because SIZE is specified, the file in not autoextensible.
- A 200 MB SYSAUX tablespace datafile in directory /u01/oradata. Because SIZE is specified, the file in not autoextensible. The tablespace is locally managed with automatic segment-space management.
- Two redo log groups with two members of 100 MB each, one each in directories /u02/oradata and /u03/oradata.
- For the default temporary tablespace dflt_ts, a 10 MB tempfile in directory /u01/oradata. Because SIZE is specified, the file in not autoextensible.
- For the undo tablespace undo_ts, a 10 MB datafile in directory /u01/oradata. Because SIZE is specified, the file in not autoextensible.
- If no CONTROL_FILES initialization parameter was specified, then two control files, one each in directories /u02/oradata and /u03/oradata. The control file in /u02/oradata is the primary control file.
The following parameter settings are included in the initialization parameter file:

DB_CREATE_FILE_DEST = '/u01/oradata'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'

The following statement is issued at the SQL prompt:

```
SQL> CREATE DATABASE sample3 DATAFILE SIZE 400M
     2>   SYSAUX DATAFILE SIZE 200M
     3>   DEFAULT TEMPORARY TABLESPACE dflt_ts TEMPFILE SIZE 10M
     4>   UNDO TABLESPACE undo_ts DATAFILE SIZE 10M;
```

11.9. Creating Datafiles for Tablespaces Using Oracle-Managed Files

The following statements that can create datafiles are relevant to the discussion in this section:

- CREATE TABLESPACE
- CREATE UNDO TABLESPACE
- ALTER TABLESPACE ... ADD DATAFILE

When creating a tablespace, either a regular tables pace or an undo tablespace, the DATAFILE clause is optional. When you include the DATAFILE clause the filename is optional. If the DATAFILE clause or filename is not provided, then the following rules apply:

- If the DB_CREATE_FILE_DEST initialization parameter is specified, then an Oracle-managed datafile is created in the location specified by the parameter.
- If the DB_CREATE_FILE_DEST initialization parameter is not specified, then the statement creating the datafile fails. When you add a datafile to a tablespace with the ALTER TABLESPACE...ADD DATAFILE statement the filename is optional. If the filename is not specified, then the same rules apply as discussed in the previous paragraph. By default, an Oracle-managed datafile for a regular tablespace is 100 MB and is autoextensible with an unlimited maximum size. However, if in your DATAFILE clause you override these defaults by specifying a SIZE value (and no AUTOEXTEND clause), then the datafile is not autoextensible.

Examples

The following are some examples of creating tablespaces with Oracle-managed files.

Example: The following example sets the default location for datafile creations to /u01/oradata and then creates a tablespace tbs_1 with a datafile in that location. The datafile is 100 MB and is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TABLESPACE tbs_1;
```

Example: This example creates a tablespace named tbs_2 with a datafile in the directory /u01/oradata. The datafile initial size is 400 MB, and because the SIZE clause is specified, the datafile is not autoextensible. The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

The following statement is issued at the SQL prompt:
SQL> CREATE TABLESPACE tbs_2 DATAFILE SIZE 400M;

**Example** : This example creates a tablespace named tbs_3 with an autoextensible datafile in the directory /u01/oradata with a maximum size of 800 MB and an initial size of 100 MB:

The following parameter setting is included in the initialization parameter file:

\[DB\_CREATE\_FILE\_DEST = '/u01/oradata'\]

The following statement is issued at the SQL prompt:

SQL> CREATE TABLESPACE tbs_3 DATAFILE AUTOEXTEND ON MAXSIZE 800M;

Example: The following example sets the default location for datafile creations to /u01/oradata and then creates a tablespace named tbs_4 in that directory with two datafiles. Both datafiles have an initial size of 200 MB, and because a SIZE value is specified, they are not autoextensible

SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';

SQL> CREATE TABLESPACE tbs_4 DATAFILE SIZE 200M SIZE 200M;

Example: The following example creates an undo tablespace named undotbs_1 with a datafile in the directory /u01/oradata. The datafile for the undo tablespace is 100 MB and is autoextensible with an unlimited maximum size. The following parameter setting is included in the initialization parameter file:

\[DB\_CREATE\_FILE\_DEST = '/u01/oradata'\]

The following statement is issued at the SQL prompt:

SQL> CREATE UNDO TABLESPACE undotbs_1;

Example

This example adds an Oracle-managed autoextensible datafile to the tbs_1 tablespace. The datafile has an initial size of 100 MB and a maximum size of 800 MB. The following parameter setting is included in the initialization parameter file:

\[DB\_CREATE\_FILE\_DEST = '/u01/oradata'\]

The following statement is entered at the SQL prompt:

SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE AUTOEXTEND ON MAXSIZE 800M;

Creating Tempfiles for Temporary Tablespaces Using Oracle-Managed Files

The following statements that create tempfiles are relevant to the discussion in this section:

- CREATE TEMPORARY TABLESPACE
- ALTER TABLESPACE ... ADD TEMPFILE

When creating a temporary tablespace the TEMPFILE clause is optional. If you include the TEMPFILE clause, then the filename is optional. If the TEMPFILE clause or filename is not provided, then the following rules apply:

- If the DB_CREATE_FILE_DEST initialization parameter is specified, then an Oracle-managed tempfile is created in the location specified by the parameter.

- If the DB_CREATE_FILE_DEST initialization parameter is not specified, then the statement creating the tempfile fails.

When you add a tempfile to a tablespace with the ALTER TABLESPACE...ADD TEMPFILE statement the filename is optional. If the filename is not specified, then the same rules apply as discussed in the previous paragraph. When
overriding the default attributes of an Oracle-managed file, if a SIZE value is specified but no AUTOEXTEND clause is specified, then the datafile is not autoextensible.

Example: The following example sets the default location for datafile creations to /u01/oradata and then creates a tablespace named temptbs_1 with a tempfile in that location. The tempfile is 100 MB and is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TEMPORARY TABLESPACE temptbs_1;
```

Example: The following example sets the default location for datafile creations to /u03/oradata and then adds a tempfile in the default location to a tablespace named temptbs_1. The tempfile initial size is 100 MB. It is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata';
SQL> ALTER TABLESPACE TBS_1 ADD TEMPFILE;
```

11.10. Creating Control Files Using Oracle-Managed Files

When you issue the CREATE CONTROLFILE statement, a control file is created (or reused, if REUSE is specified) in the files specified by the CONTROL_FILES initialization parameter. If the CONTROL_FILES parameter is not set, then the control file is created in the default control file destinations. If Oracle Database creates an Oracle-managed control file, and there is a server parameter file, then the database creates a CONTROL_FILES initialization parameter for the server parameter file. If there is no server parameter file, then you must create a CONTROL_FILES initialization parameter manually and include it in the initialization parameter file. If the datafiles in the database are Oracle-managed files, then the database-generated filenames for the files must be supplied in the DATAFILE clause of the statement. If the redo log files are Oracle-managed files, then the NORESETLOGS or RESETLOGS keyword determines what can be supplied in the LOGFILE clause:

- If the NORESETLOGS keyword is used, then the database-generated filenames for the Oracle-managed redo log files must be supplied in the LOGFILE clause.
- If the RESETLOGS keyword is used, then the redo log file names can be supplied as with the CREATE DATABASE statement.

The sections that follow contain examples of using the CREATE CONTROLFILE statement with Oracle-managed files.

Example: The following CREATE CONTROLFILE statement is generated by an ALTER DATABASE BACKUP CONTROLFILE TO TRACE statement for a database with Oracle-managed datafiles and redo log files:

```
CREATE CONTROLFILE
  DATABASE sample
  LOGFILE
    GROUP 1 ('/u01/oradata/SAMPLE/online/log/ol_mf_1_o220rtt9_.log',
             '/u02/oradata/SAMPLE/online/log/ol_mf_1_v2o0b2i3_.log')
             SIZE 100M,
    GROUP 2 ('/u01/oradata/SAMPLE/online/log/ol_mf_2_p22056iw_.log',
             '/u02/oradata/SAMPLE/online/log/ol_mf_2_p02rcyg3_.log')
             SIZE 100M
  NORESETLOGS
  DATAFILE '/u01/oradata/SAMPLE/datafile/ol_mf_system_xu34yb2_.dbf'
             SIZE 100M,
          '/u01/oradata/SAMPLE/datafile/ol_mf_sysaux_aawbmz51_.dbf'
             SIZE 100M,
          '/u01/oradata/SAMPLE/datafile/ol_mf_sys_undotbs_apqbzm51_.dbf'
             SIZE 100M
```
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG;

CREATE CONTROLFILE Using RESETLOGS Keyword: Example
The following is an example of a CREATE CONTROLFILE statement with the RESETLOGS option. Some combination of DB_CREATE_FILE_DEST, DB_RECOVERY_FILE_DEST, and DB_CREATE_ONLINE_LOG_DEST_n or must be set.

CREATE CONTROLFILE USING RESETLOGS
DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_aawbmz51_.dbf',
'/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_axybmz51_.dbf',
'/u01/oradata/SAMPLE/datafile/o1_mf_sys_undotbs_aazzbmz51_.dbf'
SIZE 100M
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG;

Later, you must issue the ALTER DATABASE OPEN RESETLOGS statement to re-create the redo log files.

11.11. Creating Redo Log Files Using Oracle-Managed Files

Redo log files are created at database creation time. They can also be created when you issue either of the following statements:

Using the ALTER DATABASE ADD LOGFILE Statement
The ALTER DATABASE ADD LOGFILE statement lets you later add a new group to your current redo log. The filename in the ADD LOGFILE clause is optional if you are using Oracle-managed files. If a filename is not provided, then a redo log file is created in the default log file destination. If a filename is not provided and you have not provided one of the initialization parameters require for creating Oracle-managed files, then the statement returns an error. The default size for an Oracle-managed log file is 100 MB. You continue to add and drop redo log file members by specifying complete filenames. Adding New Redo Log Files: Example The following example creates a log group with a member in /u01/oradata and another member in /u02/oradata. The size of each log file is 100 MB. The following parameter settings are included in the initialization parameter file:

DB_CREATE_ONLINE_LOG_DEST_1 = '/u01/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u02/oradata'

The following statement is issued at the SQL prompt:

SQL> ALTER DATABASE ADD LOGFILE;

Using the ALTER DATABASE OPEN RESETLOGS Statement
If you previously created a control file specifying RESETLOGS and either did not specify filenames or specified nonexistent filenames, then the database creates redo log files for you when you issue the ALTER DATABASE OPEN RESETLOGS statement.

11.12. Creating Archived Logs Using Oracle-Managed Files

Archived logs are created in the DB_RECOVERY_FILE_DEST location when:
• The ARC or LGWR background process archives an online redo log or
• An ALTER SYSTEM ARCHIVE LOG CURRENT statement is issued.

For example, assume that the following parameter settings are included in the initialization parameter file:

```sql
DB_RECOVERY_FILE_DEST_SIZE = 20G
DB_RECOVERY_FILE_DEST = '/u01/oradata'
LOG_ARCHIVE_DEST_1 = 'LOCATION=USE_DB_RECOVERY_FILE_DEST'
```

### 11.13. Behavior of Oracle-Managed Files

The filenames of Oracle-managed files are accepted in SQL statements wherever a filename is used to identify a non-existing file. These filenames, like other filenames, are stored in the control file and, if using Recovery Manager (RMAN) for backup and recovery, in the RMAN catalog. They are visible in all of the usual fixed and dynamic performance views that are available for monitoring datafiles and tempfiles (for example, V$DATAFILE or DBA_DATA_FILES). The following are some examples of statements using database-generated filenames:

```sql
SQL> ALTER DATABASE
  2> RENAME FILE '/u01/oradata/mydb/datafile/o1_mf_tbs01_ziw3bopb_.dbf'
  3> TO '/u01/oradata/mydb/tbs0101.dbf';
SQL> ALTER DATABASE
  2> DROP LOGFILE '/u01/oradata/mydb/onlinelog/o1_mf_1_wo94n2xi_.log';
SQL> ALTER TABLE emp
  2> ALLOCATE EXTENT
  3> (DATAFILE '/u01/oradata/mydb/datafile/o1_mf_tbs1_2ixfh90q_.dbf');
```

You can backup and restore Oracle-managed datafiles, tempfiles, and control files as you would corresponding non-Oracle-managed files. Using database-generated filenames does not impact the use of logical backup files such as export files. This is particularly important for tablespace point-in-time recovery (TSPITR) and transportable tablespace export files. There are some cases where Oracle-managed files behave differently. These are discussed in the sections that follow.

### 11.14. Dropping Datafiles and Tempfiles

Unlike files that are not managed by the database, when a non-Oracle-managed datafile or tempfile is dropped, the filename is removed from the control file and the file is automatically deleted from the file system. The statements that delete Oracle-managed files when they are dropped are:

• DROP TABLESPACE
• ALTER DATABASE TEMPFILE ... DROP

You can also use these statements, which always delete files, Oracle-managed or not:

• ALTER TABLESPACE ... DROP DATAFILE
• ALTER TABLESPACE ... DROP TEMPFILE

### 11.15. Dropping Redo Log Files

When an Oracle-managed redo log file is dropped its Oracle-managed files are deleted. You specify the group or members to be dropped. The following statements drop and delete redo log files:

• ALTER DATABASE DROP LOGFILE
11.16. Renaming Files

The following statements are used to rename files:

- `ALTER DATABASE RENAME FILE`
- `ALTER TABLESPACE ... RENAME DATAFILE`

These statements do not actually rename the files on the operating system, but rather, the names in the control file are changed. If the old file is an Oracle-managed file and it exists, then it is deleted. You must specify each filename using the conventions for filenames on your operating system when you issue this statement.

11.17. Scenarios for Using Oracle-Managed Files

This section further demonstrates the use of Oracle-managed files by presenting scenarios of their use.

**Scenario 1:** Create and Manage a Database with Multiplexed Redo Logs

In this scenario, a DBA creates a database where the datafiles and redo log files are created in separate directories. The redo log files and control files are multiplexed. The database uses an undo tablespace, and has a default temporary tablespace. The following are tasks involved with creating and maintaining this database.

1. Setting the initialization parameters
   
The DBA includes three generic file creation defaults in the initialization parameter file before creating the database. Automatic undo management mode is also specified.

   ```
   DB_CREATE_FILE_DEST = '/u01/oradata'
   DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
   DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
   UNDO_MANAGEMENT = AUTO
   ```

   The `DB_CREATE_FILE_DEST` parameter sets the default file system directory for the datafiles and tempfiles. The `DB_CREATE_ONLINE_LOG_DEST_1` and `DB_CREATE_ONLINE_LOG_DEST_2` parameters set the default file system directories for redo log file and control file creation. Each redo log file and control file is multiplexed across the two directories.

2. Creating a database
   
   Once the initialization parameters are set, the database can be created by using this statement:

   ```sql
   SQL> CREATE DATABASE sample
   2>   DEFAULT TEMPORARY TABLESPACE dflttmp;
   ```

   Because a `DATAFILE` clause is not present and the `DB_CREATE_FILE_DEST` initialization parameter is set, the `SYSTEM` tablespace datafile is created in the default file system (`/u01/oradata` in this scenario). The filename is uniquely generated by the database. The file is autoextensible with an initial size of 100 MB and an unlimited maximum size. The file is an Oracle-managed file. A similar datafile is created for the `SYSAUX` tablespace. Because a `LOGFILE` clause is not present; two redo log groups are created. Each log group has two members, with one member in the `DB_CREATE_ONLINE_LOG_DEST_1` location and the other member in the `DB_CREATE_ONLINE_LOG_DEST_2` location. The filenames are uniquely generated by the database. The log files are created with a size of 100 MB. The log file members are Oracle-managed files.

   Similarly, because the `CONTROL_FILES` initialization parameter is not present, and two `DB_CREATE_ONLINE_LOGDEST_n` initialization parameters are specified, two control files are created. The control file located in the `DB_CREATE_ONLINE_LOG_DEST_1` location is the primary control file; the control file located in the `DB_CREATE_ONLINE_LOG_DEST_2` location is a multiplexed copy. The filenames are uniquely generated by the database. They are Oracle-managed files. Assuming there is a server parameter file, a `CONTROL_FILES` initialization parameter is generated.

   Automatic undo management mode is specified, but because an undo tablespace is not specified and the
DB_CREATE_FILE_DEST initialization parameter is set, a default undo tablespace named SYS_UNDOTBS is created in the directory specified by DB_CREATE_FILE_DEST. The datafile is a 10 MB datafile that is autoextensible. It is an Oracle-managed file. Lastly, a default temporary tablespace named dflttmp is specified.

Because DB_CREATE_FILE_DEST is included in the parameter file, the tempfile for dflttmp is created in the directory specified by that parameter. The tempfile is 100 MB and is autoextensible with an unlimited maximum size. It is an Oracle-managed file. The internally generated filenames can be seen when selecting from the usual views. For example:

```sql
SQL> SELECT NAME FROM V$DATAFILE;
NAME
------------------------------
/u01/oradata/SAMPLE/datafile/o1_mf_system_cmr7t30p_.dbf
/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_cmr7t88p_.dbf
/u01/oradata/SAMPLE/datafile/o1_mf_sys_undotbs_2ixfh90q_.dbf
3 rows selected
```

3. Managing control files

The control file was created when generating the database and a CONTROL_FILES initialization parameter was added to the parameter file. If needed, then the DBA can re-create the control file or build a new one for the database using the CREATE CONTROLFILE statement. The correct Oracle-managed filenames must be used in the DATAFILE and LOGFILE clauses. The ALTER DATABASE BACKUP CONTROLFILE TO TRACE statement generates a script with the correct filenames. Alternatively, the filenames can be found by selecting from the V$DATAFILE, V$TEMPFILE, and V$LOGFILE views. The following example re-creates the control file for the sample database:

```
CREATE CONTROLFILE REUSE
DATABASE sample
LOGFILE
    GROUP 1('/u02/oradata/SAMPLE/online/log/o1_mf_1_0orm31z_.log',
            '/u03/oradata/SAMPLE/online/log/o1_mf_1_ixfvm8w9_.log'),
    GROUP 2('/u02/oradata/SAMPLE/online/log/o1_mf_2_2xyz16am_.log',
            '/u03/oradata/SAMPLE/online/log/o1_mf_2_q89tmp28_.log')
NORESETLOGS
DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_cmr7t30p_.dbf',
    '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_cmr7t88p_.dbf',
    '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undotbs_2ixfh90q_.dbf',
    '/u01/oradata/SAMPLE/datafile/o1_mf_dflttmp_157se6ff_.tmp'
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG;
```

The control file created by this statement is located as specified by the CONTROL_FILES initialization parameter that was generated when the database was created. The REUSE clause causes any existing files to be overwritten.

4. Managing the redo log

To create a new group of redo log files, the DBA can use the ALTER DATABASE ADD LOGFILE statement. The following statement adds a log file with a member in the DB_CREATE_ONLINE_LOG_DEST_1 location and a member in the DB_CREATE_ONLINE_LOG_DEST_2 location. These files are Oracle-managed files.

```sql
SQL> ALTER DATABASE ADD LOGFILE;
```

Log file members continue to be added and dropped by specifying complete filenames. The GROUP clause can be used to drop a log group. In the following example the operating system file associated with each Oracle-managed log file member is automatically deleted.

```sql
SQL> ALTER DATABASE DROP LOGFILE GROUP 3;
```
5. Managing tablespaces
The default storage for all datafiles for future tablespace creations in the sample database is the location specified
by the DB_CREATE_FILE_DEST initialization parameter (/u01/oradata in this scenario). Any datafiles for which no
filename is specified, are created in the file system specified by the initialization parameter
DB_CREATE_FILE_DEST. For example:

SQL> CREATE TABLESPACE tbs_1;

The preceding statement creates a tablespace whose storage is in /u01/oradata. Datafile is created with an initial of
100 MB and it is autoextensible with an unlimited maximum size. The datafile is an Oracle-managed file. When the
tablespace is dropped, the Oracle-managed files for the tablespace are automatically removed. The following
statement drops the tablespace and all the Oracle-managed files used for its storage:

SQL> DROP TABLESPACE tbs_1;

Once the first datafile is full, the database does not automatically create a new datafile. More space can be added to
the tablespace by adding another Oracle-managed datafile. The following statement adds another datafile in the
location specified by DB_CREATE_FILE_DEST:

SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE;

The default file system can be changed by changing the initialization parameter. This does not change any existing
datafiles. It only affects future creations. This can be done dynamically using the following statement:

SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST='/u04/oradata';

6. Archiving redo information
Archiving of redo log files is no different for Oracle-managed files, than it is for unmanaged files. A file system
location for the archived log files can be specified using the LOG_ARCHIVE_DEST_n initialization parameters. The
filenames are formed based on the LOG_ARCHIVE_FORMAT parameter or its default. The archived logs are not
Oracle-managed files
7. Backup, restore, and recover
Since an Oracle-managed file is compatible with standard operating system files, you can use operating system
utilities to backup or restore Oracle-managed files.

Scenario 2: Create and Manage a Database with Database and Flash Recovery Areas
In this scenario, a DBA creates a database where the control files and redo log files are multiplexed. Archived logs
and RMAN backups are created in the flash recovery area. The following tasks are involved in creating and
maintaining this database:
1. Setting the initialization parameters
The DBA includes the following generic file creation defaults:

DB_CREATE_FILE_DEST = '/u01/oradata'
DB_RECOVERY_FILE_DEST_SIZE = 10G
DB_RECOVERY_FILE_DEST = '/u02/oradata'
LOG_ARCHIVE_DEST_1 = 'LOCATION = USE_DB_RECOVERY_FILE_DEST'

The DB_CREATE_FILE_DEST parameter sets the default file system directory for datafiles, tempfiles, control files,
and redo logs. The DB_RECOVERY_FILE_DEST parameter sets the default file system directory for control files,
redo logs, and RMAN backups. The LOG_ARCHIVE_DEST_1 configuration
'LOCATION=USE_DB_RECOVERY_FILE_DEST' redirects archived logs to the DB_RECOVERY_FILE_DEST
location. The DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST parameters set the default directory for
log file and control file creation. Each redo log and control file is multiplexed across the two directories.
2. Creating a database
3. Managing control files
4. Managing the redo log
5. Managing tablespaces
Tasks 2, 3, 4, and 5 are the same as in Scenario 1, except that the control files and redo logs are multiplexed across the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST locations.

6. Archiving redo log information
Archiving online logs is no different for Oracle-managed files than it is for unmanaged files. The archived logs are created in DB_RECOVERY_FILE_DEST and are Oracle-managed files.

7. Backup, restore, and recover
An Oracle-managed file is compatible with standard operating system files, so you can use operating system utilities to backup or restore Oracle-managed files. All existing methods for backing up, restoring, and recovering the database work for Oracle-managed files. When no format option is specified, all disk backups by RMAN are created in the DB_RECOVERY_FILE_DEST location. The backups are Oracle-managed files.

**Scenario 3: Adding Oracle-Managed Files to an Existing Database**
Assume in this case that an existing database does not have any Oracle-managed files, but the DBA would like to create new tablespaces with Oracle-managed files and locate them in directory /u03/oradata.

1. Setting the initialization parameters
To allow automatic datafile creation, set the DB_CREATE_FILE_DEST initialization parameter to the file system directory in which to create the datafiles. This can be done dynamically as follows:

```sql
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata';
```

2. Creating tablespaces
Once DB_CREATE_FILE_DEST is set, the DATAFILE clause can be omitted from a CREATE TABLESPACE statement. The datafile is created in the location specified by DB_CREATE_FILE_DEST by default. For example:

```sql
SQL> CREATE TABLESPACE tbs_2;
```

When the tbs_2 tablespace is dropped, its datafiles are automatically deleted.
12. Tables Management

12.1. About Tables

Tables are the basic unit of data storage in an Oracle Database. Data is stored in rows and columns. You define a table with a table name, such as employees, and a set of columns. You give each column a column name, such as employee_id, last_name, and job_id; a datatype, such as VARCHAR2, DATE, or NUMBER; and a width. The width can be predetermined by the datatype, as in DATE. If columns are of the NUMBER datatype, define precision and scale instead of width. A row is a collection of column information corresponding to a single record. You can specify rules for each column of a table. These rules are called integrity constraints. One example is a NOT NULL integrity constraint. This constraint forces the column to contain a value in every row.

You can invoke transparent data encryption to encrypt data before storing it. If users attempt to circumvent the database access control mechanisms by looking inside Oracle datafiles directly with operating system tools, encryption prevents these users from viewing sensitive data. Tables can also include virtual columns. A virtual column is like any other table column, except that its value is derived by evaluating an expression. The expression can include columns from the same table, constants, SQL functions, and user-defined PL/SQL functions. You cannot explicitly write to a virtual column. After you create a table, you insert rows of data using SQL statements or using an Oracle bulk load utility. Table data can then be queried, deleted, or updated using SQL.

12.2. Guidelines for Managing Tables

This section describes guidelines to follow when managing tables. Following these guidelines can make the management of your tables easier and can improve performance when creating the table, as well as when loading, updating, and querying the table data.

12.3. Design Tables before Creating Them

Usually, the application developer is responsible for designing the elements of an application, including the tables. Database administrators are responsible for establishing the attributes of the underlying tablespace that will hold the application tables. Either the DBA or the applications developer, or both working jointly, can be responsible for the actual creation of the tables, depending upon the practices for a site. Working with the application developer, consider the following guidelines when designing tables:

- Use descriptive names for tables, columns, indexes, and clusters.
- Be consistent in abbreviations and in the use of singular and plural forms of table names and columns.
- Document the meaning of each table and its columns with the COMMENT command.
- Normalize each table.
- Select the appropriate datatype for each column.
- Consider whether your applications would benefit from adding one or more virtual columns to some tables.
- Define columns that allow nulls last, to conserve storage space.
- Cluster tables whenever appropriate, to conserve storage space and optimize performance of SQL statements.
- Before creating a table, you should also determine whether to use integrity constraints. Integrity constraints can be defined on the columns of a table to enforce the business rules of your database automatically.
- Specify the Location of Each Table

It is advisable to specify the TABLESPACE clause in a CREATE TABLE statement to identify the tablespace that is to store the new table. Ensure that you have the appropriate privileges and quota on any tablespaces that you use. If you do not specify a tablespace in a CREATE TABLE statement, the table is created in your default tablespace. When specifying the tablespace to contain a new table, ensure that you understand implications of your selection. By
properly specifying a tablespace during the creation of each table, you can increase the performance of the database system and decrease the time needed for database administration. The following situations illustrate how not specifying a tablespace, or specifying an inappropriate one, can affect performance:

If users' objects are created in the SYSTEM tablespace, the performance of the database can suffer, since both data dictionary objects and user objects must contend for the same datafiles. Users' objects should not be stored in the SYSTEM tablespace. To avoid this, ensure that all users are assigned default tablespaces when they are created in the database. If application-associated tables are arbitrarily stored in various tablespaces, the time necessary to complete administrative operations (such as backup and recovery) for the data of that application can be increased.

12.4. Consider Parallelizing Table Creation

You can utilize parallel execution when creating tables using a subquery (AS SELECT) in the CREATE TABLE statement. Because multiple processes work together to create the table, performance of the table creation operation is improved.

12.5. Consider Using NOLOGGING When Creating Tables

To create a table most efficiently use the NOLOGGING clause in the CREATE TABLE...AS SELECT statement. The NOLOGGING clause causes minimal redo information to be generated during the table creation. This has the following benefits:

- Space is saved in the redo log files.
- The time it takes to create the table is decreased.
- Performance improves for parallel creation of large tables.

The NOLOGGING clause also specifies that subsequent direct loads using SQL*Loader and direct load INSERT operations are not logged. Subsequent DML statements (UPDATE, DELETE, and conventional path insert) are unaffected by the NOLOGGING attribute of the table and generate redo. If you cannot afford to lose the table after you have created it (for example, you will no longer have access to the data used to create the table) you should take a backup immediately after the table is created. In some situations, such as for tables that are created for temporary use, this precaution may not be necessary. In general, the relative performance improvement of specifying NOLOGGING is greater for larger tables than for smaller tables. For small tables, NOLOGGING has little effect on the time it takes to create a table. However, for larger tables the performance improvement can be significant, especially when you are also parallelizing the table creation.

12.6. Consider Using Table Compression

As your database grows in size to gigabytes or terabytes and beyond, consider using table compression. Table compression saves disk space and reduces memory use in the buffer cache. Table compression can also speed up query execution during reads. There is, however, a cost in CPU overhead for data loading and DML. Table compression is completely transparent to applications. It is especially useful in online analytical processing (OLAP) systems, where there are lengthy read-only operations, but can also be used in online transaction processing (OLTP) systems. You specify table compression with the COMPRESS clause of the CREATE TABLE statement. You can enable compression for an existing table by using this clause in an ALTER TABLE statement. In this case, the only data that is compressed is the data inserted or updated after compression is enabled. Similarly, you can disable table compression for an existing compressed table with the ALTER TABLE...NOCOMPRESS statement. In this case, all data the was already compressed remains compressed, and new data is inserted uncompressed.

You can enable compression for all table operations or you can enable it for direct-path inserts only. When compression is enabled for all operations, compression occurs during all DML statements and when data is inserted with a bulk (direct-path) insert operation. To enable compression for conventional DML, you must set the COMPATIBLE initialization parameter to 11.1.0 or higher. To enable compression for all operations you must use the COMPRESS FOR ALL OPERATIONS clause. To enable compression for direct-path inserts only, you use the COMPRESS FOR DIRECT_LOAD_OPERATIONS clause. The keyword COMPRESS by itself is the same as the clause COMPRESS FOR DIRECT_LOAD OPERATIONS, and invokes the same compression behavior as previous
12.6.1. Adding and Dropping Columns in Compressed Tables

When you enable compression for all operations on a table, you can add and drop table columns. If you enable compression for direct-path inserts only, you cannot drop columns, and you can add columns only if you do not specify default values.

Examples

The following example enables compression for all operations on the table transaction, which is used in an OLTP application:

```
CREATE TABLE transaction ( ... ) COMPRESS FOR ALL OPERATIONS;
```

The next two examples enable compression for direct-path insert only on the sales_history table, which is a fact table in a data warehouse:

```
CREATE TABLE sales_history ( ... ) COMPRESS FOR DIRECT_LOAD OPERATIONS;
CREATE TABLE sales_history ( ... ) COMPRESS;
```

12.6.2. Compression and Partitioned Tables

You can enable or disable compression at the partition level. You can therefore have a table with both compressed and uncompressed partitions. If the compression settings for a table and one of its partitions disagree, the partition setting has precedence for the partition. In the following example, all partitions except the northeast partition are compressed.

```
CREATE TABLE sales
  (saleskey number,
   quarter number,
   product number,
   salesperson number,
   amount number(12, 2),
   region varchar2(10)) COMPRESS
PARTITION BY LIST (region)
  (PARTITION northwest VALUES ('NORTHWEST'),
   PARTITION southwest VALUES ('SOUTHWEST'),
   PARTITION northeast VALUES ('NORTHEAST') NOCOMPRESS,
   PARTITION southeast VALUES ('SOUTHEAST'),
   PARTITION western VALUES ('WESTERN'));
```

12.6.3. Determining If a Table is compressed

In the * _TABLES data dictionary views, compressed tables have ENABLED in the COMPRESSION column. For partitioned tables, this column is null, and the COMPRESSION column of the * _TAB_PARTITIONS data dictionary view indicates the partitions that are compressed. In addition, the COMPRESS_FOR column indicates whether the table is compressed FOR ALL OPERATIONS or for DIRECT LOAD ONLY.

```
SQL> SELECT table_name, compression, compress_for FROM user_tables;

TABLE_NAME       COMPRESS COMPRESS_FOR
----------------- ------------- ------------
T1                DISABLED
T2                ENABLED DIRECT LOAD ONLY
T3                ENABLED FOR ALL OPERATIONS
```
12.7 Consider Encrypting Columns That Contain Sensitive Data

You can encrypt individual table columns that contain sensitive data. Examples of sensitive data include social security numbers, credit card numbers, and medical records. Column encryption is transparent to your applications, with some restrictions. Although encryption is not meant to solve all security problems, it does protect your data from users who try to circumvent the security features of the database and access database files directly through the operating system file system. Column encryption uses the transparent data encryption feature of Oracle Database, which requires that you create an Oracle wallet to store the master encryption key for the database. The wallet must be open before you can create a table with encrypted columns and before you can store or retrieve encrypted data. When you open the wallet, it is available to all sessions, and it remains open until you explicitly close it or until the database is shut down.

Transparent data encryption supports industry-standard encryption algorithms, including the following Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) algorithms:

- 3DES168
- AES128
- AES192
- AES256

You choose the algorithm to use when you create the table. All encrypted columns in the table use the same algorithm. The default is AES192. The encryption key length is implied by the algorithm name. For example, the AES128 algorithm uses 128-bit keys. If you plan on encrypting many columns in one or more tables, you may want to consider encrypting an entire tablespace instead and storing these tables in that tablespace. Tablespace encryption, which also uses the transparent data encryption feature but encrypts at the physical block level, can perform better than encrypting many columns. Another reason to encrypt at the tablespace level is to address the following limitations of column encryption:

- If the COMPATIBLE initialization parameter set to 10.2.0, which is the minimum setting to enable transparent data encryption, data from encrypted columns that is involved in a sort or hash-join and that must be written to a temporary tablespace is written in clear text, and thus exposed to attacks. You must set COMPATIBLE to 11.1.0 or higher to ensure that encrypted data written to a temporary tablespace remains encrypted. Note that as long as COMPATIBLE is set to 10.2.0 or higher, data from encrypted columns remains encrypted when written to the undo tablespace or the redo log.
- Certain data types, such as object data types, are not supported for column encryption. You cannot use the transportable tablespace feature for a tablespace that includes tables with encrypted columns.

12.7. Estimate Table Size and Plan Accordingly

Estimate the sizes of tables before creating them. Preferably, do this as part of database planning. Knowing the sizes, and uses, for database tables is an important part of database planning. You can use the combined estimated size of tables, along with estimates for indexes, undo space, and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases. You can use the estimated size and growth rate of an individual table to better determine the attributes of a tablespace and its underlying datafiles that are best suited for the table. This can enable you to more easily manage the table disk space and improve I/O performance of applications that use the table.

12.8. Restrictions to Consider When Creating Tables

Here are some restrictions that may affect your table planning and usage:

- Tables containing object types cannot be imported into a pre-Oracle8 database.
• You cannot merge an exported table into a preexisting table having the same name in a different schema.
• You cannot move types and extent tables to a different schema when the original data still exists in the database.
• Oracle Database has a limit on the total number of columns that a table (or attributes that an object type) can have.

Further, when you create a table that contains user-defined type data, the database maps columns of user-defined type to relational columns for storing the user-defined type data. This causes additional relational columns to be created. This results in "hidden" relational columns that are not visible in a DESCRIBE table statement and are not returned by a SELECT * statement. Therefore, when you create an object table, or a relational table with columns of REF, varray, nested table, or object type, be aware that the total number of columns that the database actually creates for the table can be more than those you specify.

12.9. Creating Tables

To create a new table in your schema, you must have the CREATE TABLE system privilege. To create a table in another user's schema, you must have the CREATE ANY TABLE system privilege. Additionally, the owner of the table must have a quota for the tablespace that contains the table, or the UNLIMITED TABLESPACE system privilege.

Create tables using the SQL statement CREATE TABLE.

Example: Creating a Table

When you issue the following statement, you create a table named admin_emp in the hr schema and store it in the admin_tbs tablespace with an initial extent size of 50K:

```sql
CREATE TABLE hr.admin_emp (
  empno      NUMBER(5) PRIMARY KEY,
  ename      VARCHAR2(15) NOT NULL,
  ssn        NUMBER(9) ENCRYPT,
  job        VARCHAR2(10),
  mgr        NUMBER(5),
  hiredate   DATE DEFAULT (sysdate),
  photo      BLOB,
  sal        NUMBER(7,2),
  hrly_rate  NUMBER(7,2) GENERATED ALWAYS AS (sal/2080),
  comm       NUMBER(7,2),
  deptno     NUMBER(3) NOT NULL
  CONSTRAINT admin_dept_fkey REFERENCES hr.departments (department_id)
) TABLESPACE admin_tbs
STORAGE ( INITIAL 50K);
```

Note the following about this example:

• Integrity constraints are defined on several columns of the table.
• Encryption is defined on one column (ssn), through the transparent data encryption feature of Oracle Database. The Oracle Wallet must therefore be open for this CREATE TABLE statement to succeed.
• The photo column is of data type BLOB, which is a member of the set of data types called large objects (LOBs). LOBs are used to store semi-structured data (such as an XML tree) and unstructured data (such as the stream of bits in a color image).
• One column is defined as a virtual column (hrly_rate). This column computes the employee’s hourly rate as the yearly salary divided by 2,080.
Creating a Temporary Table
Temporary tables are useful in applications where a result set is to be buffered (temporarily persisted), perhaps because it is constructed by running multiple DML operations. For example, consider the following:

A Web-based airlines reservations application allows a customer to create several optional itineraries. Each itinerary is represented by a row in a temporary table. The application updates the rows to reflect changes in the itineraries. When the customer decides which itinerary she wants to use, the application moves the row for that itinerary to a persistent table. During the session, the itinerary data is private. At the end of the session, the optional itineraries are dropped. The definition of a temporary table is visible to all sessions, but the data in a temporary table is visible only to the session that inserts the data into the table. Use the CREATE GLOBAL TEMPORARY TABLE statement to create a temporary table. The ON COMMIT clause indicates if the data in the table is transaction-specific (the default) or session-specific, the implications of which are as follows:

ON COMMIT Setting Implications
DELETE ROWS This creates a temporary table that is transaction specific. A session becomes bound to the temporary table with a transactions first insert into the table. The binding goes away at the end of the transaction. The database truncates the table (delete all rows) after each commit. PRESERVE ROWS This creates a temporary table that is session specific. A session gets bound to the temporary table with the first insert into the table in the session. This binding goes away at the end of the session or by issuing a TRUNCATE of the table in the session. The database truncates the table when you terminate the session.

This statement creates a temporary table that is transaction specific:

```
CREATE GLOBAL TEMPORARY TABLE admin_work_area
    (startdate DATE,
     enddate DATE,
     class CHAR(20))
ON COMMIT DELETE ROWS;
```

Indexes can be created on temporary tables. They are also temporary and the data in the index has the same session or transaction scope as the data in the underlying table. By default, rows in a temporary table are stored in the default temporary tablespace of the user who creates it. However, you can assign a temporary table to another tablespace upon creation of the temporary table by using the TABLESPACE clause of CREATE GLOBAL TEMPORARY TABLE. You can use this feature to conserve space used by temporary tables. For example, if you need to perform many small temporary table operations and the default temporary tablespace is configured for sort operations and thus uses a large extent size, these small operations will consume lots of unnecessary disk space. In this case it is better to allocate a second temporary tablespace with a smaller extent size. The following two statements create a temporary tablespace with a 64 KB extent size, and then a new temporary table in that tablespace.

```
CREATE TEMPORARY TABLESPACE tbs_t1
    TEMPFILE 'tbs_t1.f' SIZE 50m REUSE AUTOEXTEND ON
    MAXSIZE UNLIMITED
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 64K;

CREATE GLOBAL TEMPORARY TABLE admin_work_area
    (startdate DATE,
     enddate DATE,
     class CHAR(20))
ON COMMIT DELETE ROWS
    TABLESPACE tbs_t1;
```

Unlike permanent tables, temporary tables and their indexes do not automatically allocate a segment when they are created. Instead, segments are allocated when the first INSERT (or CREATE TABLE AS SELECT) is performed. This means that if a SELECT, UPDATE, or DELETE is performed before the first INSERT, the table appears to be empty. DDL operations (except TRUNCATE) are allowed on an existing temporary table only if no session is currently bound to that temporary table. If you rollback a transaction, the data you entered is lost, although the table definition persists. A transaction-specific temporary table allows only one transaction at a time. If there are several autonomous
transactions in a single transaction scope, each autonomous transaction can use the table only as soon as the previous one commits. Because the data in a temporary table is, by definition, temporary, backup and recovery of temporary table data is not available in the event of a system failure. To prepare for such a failure, you should develop alternative methods for preserving temporary table data.

### 12.9.1 Parallelizing Table Creation

When you specify the AS SELECT clause to create a table and populate it with data from another table, you can utilize parallel execution. The CREATE TABLE...AS SELECT statement contains two parts: a CREATE part (DDL) and a SELECT part (query). Oracle Database can parallelize both parts of the statement. The CREATE part is parallelized if one of the following is true:

- A PARALLEL clause is included in the CREATE TABLE...AS SELECT statement
- An ALTER SESSION FORCE PARALLEL DDL statement is specified

The query part is parallelized if all of the following are true:

- The query includes a parallel hint specification (PARALLEL or PARALLEL_INDEX) or the CREATE part includes the PARALLEL clause or the schema objects referred to in the query have a PARALLEL declaration associated with them.
- At least one of the tables specified in the query requires either a full table scan or an index range scan spanning multiple partitions.
- If you parallelize the creation of a table that table then has a parallel declaration (the PARALLEL clause) associated with it. Any subsequent DML or queries on the table, for which parallelization is possible, will attempt to use parallel execution.

The following simple statement parallelizes the creation of a table and stores the result in a compressed format, using table compression:

```sql
CREATE TABLE hr.admin_emp_dept PARALLEL COMPRESS AS SELECT * FROM hr.employees WHERE department_id = 10;
```

In this case, the PARALLEL clause tells the database to select an optimum number of parallel execution servers when creating the table.

### Inserting Data with DML Error Logging

When you load a table using an INSERT statement with subquery, if an error occurs, the statement is terminated and rolled back in its entirety. This can be wasteful of time and system resources. For such INSERT statements, you can avoid this situation by using the DML error logging feature. To use DML error logging, you add a statement clause that specifies the name of an error logging table into which the database records errors encountered during DML operations. When you add this error logging clause to the INSERT statement, certain types of errors no longer terminate and roll back the statement. Instead, each error is logged and the statement continues. You then take corrective action on the erroneous rows at a later time. DML error logging works with INSERT, UPDATE, MERGE, and DELETE statements. This section focuses on INSERT statements. To insert data with DML error logging:

1. Create an error logging table. (Optional)
   You can create the table manually or use the DBMS_ERRLOG package to automatically create it for you.
2. Execute an INSERT statement and include an error logging clause. This clause:
   Optionally references the error logging table that you created. If you do not provide an error logging table name, the database logs to an error logging table with a default name. The default error logging table name is ERR$ followed by the first 25 characters of the name of the table that is being inserted into.
   Optionally includes a tag (a numeric or string literal in parentheses) that gets added to the error log to help identify the statement that caused the errors. If the tag is omitted, a NULL value is used.
Optionally includes a REJECT LIMIT subclause.
This subclause indicates the maximum number of errors that can be encountered before the INSERT statement terminates and rolls back. You can also specify UNLIMITED. The default reject limit is zero, which means that upon encountering the first error, the error is logged and the statement rolls back. For parallel DML operations, the reject limit is applied to each parallel server.

Note: If the statement exceeds the reject limit and rolls back, the error logging table retains the log entries recorded so far.

3. Query the error logging table and take corrective action for the rows that generated errors.

Example The following statement inserts rows into the DW_EMP table and logs errors to the ERR_EMP table. The tag 'daily_load' is copied to each log entry. The statement terminates and rolls back if the number of errors exceeds 25.

```sql
INSERT INTO dw_empl
    SELECT employee_id, first_name, last_name, hire_date, salary, department_id
    FROM employees
    WHERE hire_date > sysdate - 7
    LOG ERRORS INTO err_empl ('daily_load') REJECT LIMIT 25
```

Error Logging Table Format
The error logging table consists of two parts:

- A mandatory set of columns that describe the error. For example, one column contains the Oracle error number.
- An optional set of columns that contain data from the row that caused the error. The column names match the column names from the table being inserted into (the "DML table").

The number of columns in this part of the error logging table can be zero, one, or more, up to the number of columns in the DML table. If a column exists in the error logging table that has the same name as a column in the DML table, the corresponding data from the offending row being inserted is written to this error logging table column. If a DML table column does not have a corresponding column in the error logging table, the column is not logged. If the error logging table contains a column with a name that does not match a DML table column, the column is ignored.

Because type conversion errors are one type of error that might occur, the data types of the optional columns in the error logging table must be types that can capture any value without data loss or conversion errors. (If the optional log columns were of the same types as the DML table columns, capturing the problematic data into the log could suffer the same data conversion problem that caused the error.) The database makes a best effort to log a meaningful value for data that causes conversion errors. If a value cannot be derived, NULL is logged for the column. An error on insertion into the error logging table causes the statement to terminate.

Creating an Error Logging Table
You can create an error logging table manually, or you can use a PL/SQL package to automatically create one for you. Creating an Error Logging Table Automatically You use the DBMS_ERRLOG package to automatically create an error logging table. The CREATE_ERROR_LOG procedure creates an error logging table with all of the mandatory error description columns plus all of the columns from the named DML table, and performs the data type mappings. The following statement creates the error logging table used in the previous example.

```sql
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG ('DW_EMP', 'ERR_EMP');
```

Creating an Error Logging Table Manually You use standard DDL to manually create the error logging table. You must include all mandatory error description columns. They can be in any order, but must be the first columns in the table.

Security: The user who issues the INSERT statement with DML error logging must have INSERT privileges on the
error logging table.

12.10. Automatically Collecting Statistics on Tables

The PL/SQL package DBMS_STATS lets you generate and manage statistics for cost-based optimization. You can use this package to gather, modify, view, export, import, and delete statistics. You can also use this package to identify or name statistics that have been gathered. Formerly, you enabled DBMS_STATS to automatically gather statistics for a table by specifying the MONITORING keyword in the CREATE (or ALTER) TABLE statement. Starting with Oracle Database 11g, the MONITORING and NOMONITORING keywords have been deprecated and statistics are collected automatically. If you do specify these keywords, they are ignored. Monitoring tracks the approximate number of INSERT, UPDATE, and DELETE operations for the table since the last time statistics were gathered. Information about how many rows are affected is maintained in the SGA, until periodically (about every three hours) SMON incorporates the data into the data dictionary. This data dictionary information is made visible through the DBA_TAB_MODIFICATIONS, ALL_TAB_MODIFICATIONS, or USER_TAB_MODIFICATIONS views. The database uses these views to identify tables with stale statistics. To disable monitoring of a table, set the STATISTICS_LEVEL initialization parameter to BASIC. Its default is TYPICAL, which enables automatic statistics collection. Automatic statistics collection and the DBMS_STATS package enable the optimizer to generate accurate execution plans.

12.11. Altering Tables

You alter a table using the ALTER TABLE statement. To alter a table, the table must be contained in your schema, or you must have either the ALTER object privilege for the table or the ALTER ANY TABLE system privilege.

Caution: Before altering a table, familiarize yourself with the consequences of doing so.
If a view, materialized view, trigger, domain index, function-based index, check constraint, function, procedure of package depends on a base table, the alteration of the base table or its columns can affect the dependent object.

12.11.1. Reasons for Using the ALTER TABLE Statement

You can use the ALTER TABLE statement to perform any of the following actions that affect a table:

- Modify physical characteristics (INITRANS or storage parameters)
- Move the table to a new segment or tablespace
- Explicitly allocate an extent or deallocate unused space
- Add, drop, or rename columns, or modify an existing column definition (datatype, length, default value, NOT NULL integrity constraint, column expression (for virtual columns), and encryption properties.)
- Modify the logging attributes of the table
- Modify the CACHE/NOCACHE attributes
- Add, modify or drop integrity constraints associated with the table
- Enable or disable integrity constraints or triggers associated with the table
- Modify the degree of parallelism for the table
- Rename a table
- Put a table in read-only mode and return it to read/write mode
- Add or modify index-organized table characteristics
- Alter the characteristics of an external table
- Add or modify LOB columns
- Add or modify object type, nested table, or varray columns

Altering Physical Attributes of a Table

When altering the transaction entry setting INITRANS of a table, note that a new setting for INITRANS applies only to data blocks subsequently allocated for the table. To better understand this transaction entry setting parameter. The storage parameters INITIAL and MINEXTENTS cannot be altered. All new settings for the other storage
parameters (for example, NEXT, PCTINCREASE) affect only extents subsequently allocated for the table. The size of the next extent allocated is determined by the current values of NEXT and PCTINCREASE, and is not based on previous values of these parameters.

Moving a Table to a New Segment or Tablespace
The ALTER TABLE...MOVE statement enables you to relocate data of a non-partitioned table or of a partition of a partitioned table into a new segment, and optionally into a different tablespace for which you have quota. This statement also lets you modify any of the storage attributes of the table or partition, including those which cannot be modified using ALTER TABLE. You can also use the ALTER TABLE...MOVE statement with a COMPRESS clause to store the new segment using table compression.

One important reason to move a table to a new tablespace (with a new datafile) is to eliminate the possibility that old versions of column data—versions left on now unused portions of the disk due to segment shrink, reorganization, or previous table moves—could be viewed by bypassing the access controls of the database (for example with an operating system utility). This is especially important with columns that you intend to modify by adding transparent data encryption.

Note: The ALTER TABLE...MOVE statement does not permit DML against the table while the statement is executing.

The following statement moves the hr.admin_emp table to a new segment, specifying new storage parameters:

```
ALTER TABLE hr.admin_emp MOVE STORAGE ( INITIAL 20K NEXT 40K MINEXTENTS 2 MAXEXTENTS 20 PCTINCREASE 0 );
```

Moving a table changes the rowids of the rows in the table. This causes indexes on the table to be marked UNUSABLE, and DML accessing the table using these indexes will receive an ORA-01502 error. The indexes on the table must be dropped or rebuilt. Likewise, any statistics for the table become invalid and new statistics should be collected after moving the table. If the table includes LOB column(s), this statement can be used to move the table along with LOB data and LOB index segments (associated with this table) which the user explicitly specifies. If not specified, the default is to not move the LOB data and LOB index segments.

Manually Allocating Storage for a Table
Oracle Database dynamically allocates additional extents for the data segment of a table, as required. However, perhaps you want to allocate an additional extent for a table explicitly. For example, in an Oracle Real Application Clusters environment, an extent of a table can be allocated explicitly for a specific instance. A new extent can be allocated for a table using the ALTER TABLE...ALLOCATE EXTENT clause. You can also explicitly deallocate unused space using the DEALLOCATE UNUSED clause of ALTER TABLE.

Modifying an Existing Column Definition
Use the ALTER TABLE...MODIFY statement to modify an existing column definition. You can modify column datatype, default value, column constraint, column expression (for virtual columns) and column encryption. You can increase the length of an existing column, or decrease it, if all existing data satisfies the new length. You can change a column from byte semantics to CHAR semantics or vice versa. You must set the initialization parameter BLANK_TRIMMING=TRUE to decrease the length of a non-empty CHAR column. If you are modifying a table to increase the length of a column of datatype CHAR, realize that this can be a time consuming operation and can require substantial additional storage, especially if the table contains many rows. This is because the CHAR value in each row must be blank-padded to satisfy the new column length.

Adding Table Columns
To add a column to an existing table, use the ALTER TABLE...ADD statement. The following statement alters the hr.admin_emp table to add a new column named bonus:

```
ALTER TABLE hr.admin_emp ADD (bonus NUMBER (7,2));
```

If a new column is added to a table, the column is initially NULL unless you specify the DEFAULT clause. When you specify a default value, the database immediately updates each row with the default value. Note that this can take some time, and that during the update, there is an exclusive DML lock on the table. For some types of tables (for example, tables without LOB columns), if you specify both a NOT NULL constraint and a default value, the
database can optimize the column add operation and greatly reduce the amount of time that the table is locked for DML. You can add a column with a NOT NULL constraint only if the table does not contain any rows, or you specify a default value.

Adding a Column to a Compressed Table
If you enable compression for all operations on a table, you can add columns to that table with or without default values. If you enable compression for direct-path inserts only, you can add columns only if you do not specify default values.

Adding a Virtual Column
If the new column is a virtual column, its value is determined by its column expression. (Note that a virtual column's value is calculated only when it is queried.)

Renaming Table Columns
Oracle Database lets you rename existing columns in a table. Use the RENAME COLUMN clause of the ALTER TABLE statement to rename a column. The new name must not conflict with the name of any existing column in the table. No other clauses are allowed in conjunction with the RENAME COLUMN clause. The following statement renames the comm column of the hr.admin_emp table.

```
ALTER TABLE hr.admin_emp RENAME COLUMN comm TO commission;
```

As noted earlier, altering a table column can invalidate dependent objects. However, when you rename a column, the database updates associated data dictionary tables to ensure that function-based indexes and check constraints remain valid.

**Note:** The RENAME TO clause of ALTER TABLE appears similar in syntax to the RENAME COLUMN clause, but is used for renaming the table itself.

Dropping Table Columns
You can drop columns that are no longer needed from a table, including an index-organized table. This provides a convenient means to free space in a database, and avoids your having to export/import data then re-create indexes and constraints. You cannot drop all columns from a table, nor can you drop columns from a table owned by SYS. Any attempt to do so results in an error.

Removing Columns from Tables
When you issue an ALTER TABLE...DROP COLUMN statement, the column descriptor and the data associated with the target column are removed from each row in the table. You can drop multiple columns with one statement. The following statements are examples of dropping columns from the hr.admin_emp table. The first statement drops only the sal column:

```
ALTER TABLE hr.admin_emp DROP COLUMN sal;
```

The next statement drops both the bonus and comm columns:

```
ALTER TABLE hr.admin_emp DROP (bonus, commission);
```

Marking Columns Unused
If you are concerned about the length of time it could take to drop column data from all of the rows in a large table, you can use the ALTER TABLE...SET UNUSED statement. This statement marks one or more columns as unused, but does not actually remove the target column data or restore the disk space occupied by these columns. However, a column that is marked as unused is not displayed in queries or data dictionary views, and its name is removed so that a new column can reuse that name. All constraints, indexes, and statistics defined on the column are also removed. To mark the hiredate and mgr columns as unused, execute the following statement:

```
ALTER TABLE hr.admin_emp SET UNUSED (hiredate, mgr);
```

You can later remove columns that are marked as unused by issuing an ALTER TABLE...DROP UNUSED COLUMNS statement. Unused columns are also removed from the target table whenever an explicit drop of any
particular column or columns of the table is issued.

The data dictionary views USER_UNUSED_COL_TABS, ALL_UNUSED_COL_TABS, or DBA_UNUSED_COL_TABS can be used to list all tables containing unused columns. The COUNT field shows the number of unused columns in the table.

```
SELECT * FROM DBA_UNUSED_COL_TABS;
```

<table>
<thead>
<tr>
<th>OWNER</th>
<th>TABLE_NAME</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td>ADMIN_EMP</td>
<td>2</td>
</tr>
</tbody>
</table>

For external tables, the SET UNUSED statement is transparently converted into an ALTER TABLE DROP COLUMN statement. Because external tables consist of metadata only in the database, the DROP COLUMN statement performs equivalently to the SET UNUSED statement.

**Removing Unused Columns**

The ALTER TABLE...DROP UNUSED COLUMNS statement is the only action allowed on unused columns. It physically removes unused columns from the table and reclams disk space. In the ALTER TABLE statement that follows, the optional clause CHECKPOINT is specified. This clause causes a checkpoint to be applied after processing the specified number of rows, in this case 250. Checkpointing cuts down on the amount of undo logs accumulated during the drop column operation to avoid a potential exhaustion of undo space.

```
ALTER TABLE hr.admin_emp DROP UNUSED COLUMNS CHECKPOINT 250;
```

**Dropping Columns in Compressed Tables**

If you enable compression for all operations on a table, you can drop table columns. If you enable compression for direct-path inserts only, you cannot drop columns.

**Placing a Table in Read-Only Mode**

You can place a table in read-only mode with the ALTER TABLE...READ ONLY statement, and return it to read/write mode with the ALTER TABLE...READ WRITE statement. An example of a table for which read-only mode makes sense is a configuration table. If your application contains configuration tables that are not modified after installation and that must not be modified by users, your application installation scripts can place these tables in read-only mode. To place a table in read-only mode, you must have the ALTER TABLE privilege on the table or the ALTER ANY TABLE privilege. In addition, the COMPATIBLE initialization parameter must be set to 11.1.0 or greater. The following example places the SALES table in read-only mode:

```
ALTER TABLE SALES READ ONLY;
```

The following example returns the table to read/write mode:

```
ALTER TABLE SALES READ WRITE;
```

When a table is in read-only mode, operations that attempt to modify table data are disallowed. The following operations are not permitted on a read-only table: All DML operations on the table or any of its partitions

- TRUNCATE TABLE
- SELECT FOR UPDATE
- ALTER TABLE ADD/MODIFY/RENAME/DROP COLUMN
- ALTER TABLE SET COLUMN UNUSED
- ALTER TABLE DROP/TRUNCATE/EXCHANGE (SUB)PARTITION
- ALTER TABLE UPGRADE INCLUDING DATA or ALTER TYPE CASCADE INCLUDING TABLE DATA for a type with read-only table dependents
- Online redefinition
- FLASHBACK TABLE
The following operations are permitted on a read-only table:

- SELECT
- CREATE/ALTER/DROP INDEX
- ALTER TABLE ADD/MODIFY/DROP/ENABLE/DISABLE CONSTRAINT
- ALTER TABLE for physical property changes
- ALTER TABLE DROP UNUSED COLUMNS
- ALTER TABLE ADD/COALESCE/MERGE/MODIFY/MOVE/RENAME/SPLIT (SUB) PARTITION
- ALTER TABLE MOVE
- ALTER TABLE ENABLE ROW MOVEMENT and ALTER TABLE SHRINK
- RENAME TABLE and ALTER TABLE RENAME TO
- DROP TABLE
- ALTER TABLE DEALLOCATE UNUSED
- ALTER TABLE ADD/DROP SUPPLEMENTAL LOG

12.12. Redefining Tables Online

In any database system, it is occasionally necessary to modify the logical or physical structure of a table to:

- Improve the performance of queries or DML
- Accommodate application changes
- Manage storage

Oracle Database provides a mechanism to make table structure modifications without significantly affecting the availability of the table. The mechanism is called online table redefinition. Redefining tables online provides a substantial increase in availability compared to traditional methods of redefining tables. When a table is redefined online, it is accessible to both queries and DML during much of the redefinition process. The table is locked in the exclusive mode only during a very small window that is independent of the size of the table and complexity of the redefinition and that is completely transparent to users. Online table redefinition requires an amount of free space that is approximately equivalent to the space used by the table being redefined. More space may be required if new columns are added. You can perform online table redefinition with the Enterprise Manager Reorganize Objects wizard or with the DBMS_REDEFI尼ITION package.

12.12.1 Features of Online Table Redefinition

Online table redefinition enables you to:

- Modify the storage parameters of a table or cluster
- Move a table or cluster to a different tablespace

Note: If it is not important to keep a table available for DML when moving it to another tablespace, you can use the simpler ALTER TABLE MOVE command.

- Add, modify, or drop one or more columns in a table or cluster
- Add or drop partitioning support (non-clustered tables only)
- Change partition structure
- Change physical properties of a single table partition, including moving it to a different tablespace in the same schema
• Change physical properties of a materialized view log or an Oracle Streams Advanced Queuing queue table
• Add support for parallel queries
• Re-create a table or cluster to reduce fragmentation

**Note:** In many cases, online segment shrink is an easier way to reduce fragmentation.

### 12.12.2. Performing Online Redefinition with DBMS_REDEFINITION

You use the DBMS_REDEFINITION package to perform online redefinition of a table.

**To redefine a table online:**

Choose the redefinition method: by key or by rowid

**By key**—Select a primary key or pseudo-primary key to use for the redefinition. Pseudo-primary keys are unique keys with all component columns having NOT NULL constraints. For this method, the versions of the tables before and after redefinition should have the same primary key columns. This is the preferred and default method of redefinition.

**By rowid**—Use this method if no key is available. In this method, a hidden column named M_ROW$$ is added to the post-redefined version of the table. It is recommended that this column be dropped or marked as unused after the redefinition is complete. If COMPATIBLE is set to 10.2.0 or higher, the final phase of redefinition automatically sets this column unused. You can then use the ALTER TABLE ... DROP UNUSED COLUMNS statement to drop it. You cannot use this method on index-organized tables. Verify that the table can be redefined online by invoking the CAN_REDEF_TABLE procedure. If the table is not a candidate for online redefinition, then this procedure raises an error indicating why the table cannot be redefined online.

Create an empty interim table (in the same schema as the table to be redefined) with all of the desired logical and physical attributes. If columns are to be dropped, do not include them in the definition of the interim table. If a column is to be added, then add the column definition to the interim table. If a column is to be modified, create it in the interim table with the properties that you want. It is not necessary to create the interim table with all the indexes, constraints, grants, and triggers of the table being redefined, because these will be defined in step 6 when you copy dependent objects. (Optional) If you are redefining a large table and want to improve the performance of the next step by running it in parallel, issue the following statements:

```sql
alter session force parallel dml parallel degree-of-parallelism;
alter session force parallel query parallel degree-of-parallelism;
```

Start the redefinition process by calling START_REDEF_TABLE, providing the following:

- The schema and table name of the table to be redefined
- The interim table name
- A column mapping string that maps the columns of table to be redefined to the columns of the interim table
- The redefinition method

Package constants are provided for specifying the redefinition method. DBMS_REDEFINITION.CONS_USE_PK is used to indicate that the redefinition should be done using primary keys or pseudo-primary keys. DBMS_REDEFINITION.CONS_USE_ROWID is used to indicate that the redefinition should be done using rowids. If this argument is omitted, the default method of redefinition (CONS_USE_PK) is assumed.

**Optionally, the columns to be used in ordering rows**

If redefining only a single partition of a partitioned table, the partition name because this process involves copying data, it may take a while. The table being redefined remains available for queries and DML during the entire process.

**Note:** If START_REDEF_TABLE fails for any reason, you must call ABORT_REDEF_TABLE, otherwise subsequent
6. Copy dependent objects (such as triggers, indexes, materialized view logs, grants, and constraints) and statistics from the table being redefined to the interim table, using one of the following two methods. Method 1 is the preferred method because it is more automatic, but there may be times that you would choose to use method 2. Method 1 also enables you to copy table statistics to the interim table.

**Method 1: Automatically Creating Dependent Objects**

Use the `COPY_TABLE_DEPENDENTS` procedure to automatically create dependent objects on the interim table. This procedure also registers the dependent objects. Registering the dependent objects enables the identities of these objects and their copied counterparts to be automatically swapped later as part of the redefinition completion process. The result is that when the redefinition is completed, the names of the dependent objects will be the same as the names of the original dependent objects.

**Method 2: Manually Creating Dependent Objects**

You can manually create dependent objects on the interim table and then register them.

**Note:** In Oracle Database Release 9i, you were required to manually create the triggers, indexes, grants, and constraints on the interim table, and there may still be situations where you want to or must do so. In such cases, any referential constraints involving the interim table (that is, the interim table is either a parent or a child table of the referential constraint) must be created disabled. When online redefinition completes, the referential constraint is automatically enabled. In addition, until the redefinition process is either completed or aborted, any trigger defined on the interim table does not execute.

7. Execute the `FINISH_REDEF_TABLE` procedure to complete the redefinition of the table. During this procedure, the original table is locked in exclusive mode for a very short time, independent of the amount of data in the original table. However, `FINISH_REDEF_TABLE` will wait for all pending DML to commit before completing the redefinition.

8. If you used rowids for the redefinition and your `COMPATIBLE` initialization parameter is set to 10.1.0 or lower, drop or set UNUSED the hidden column `M_ROW$$` that is now in the redefined table.

```
ALTER TABLE table_name SET UNUSED (M_ROW$$);
```

If `COMPATIBLE` is 10.2.0 or higher, this hidden column is automatically set UNUSED when redefinition completes. You can then drop the column with the `ALTER TABLE ... DROP UNUSED COLUMNS` statement.

9. Wait for any long-running queries against the interim table to complete, and then drop the interim table. If you drop the interim table while there are active queries running against it, you may encounter an ORA-08103 error ("object no longer exists").

The column mapping string that you pass as an argument to `START_REDEF_TABLE` contains a comma-separated list of column mapping pairs, where each pair has the following syntax:

```
[expression]  column_name
```

The `column_name` term indicates a column in the interim table. The optional expression can include columns from the table being redefined, constants, operators, function or method calls, and so on, in accordance with the rules for expressions in a SQL SELECT statement. However, only simple deterministic subexpressions—that is, subexpressions whose results do not vary between one evaluation and the next—plus sequences and `SYSDATE` can be used. No subqueries are permitted. In the simplest case, the expression consists of just a column name from the table being redefined. If an expression is present, its value is placed in the designated interim table column during redefinition. If the expression is omitted, it is assumed that both the table being redefined and the interim table have a column named `column_name`, and the value of that column in the table being redefined is placed in the same column in the interim table.

For example, if the override column in the table being redefined is to be renamed to `override_commission`, and every override commission is to be raised by 2%, the correct column mapping pair is:

```
override*1.02  override_commission
```

If you supply '*' or NULL as the column mapping string, it is assumed that all the columns (with their names
unchanged) are to be included in the interim table. Otherwise, only those columns specified explicitly in the string are considered. The order of the column mapping pairs is unimportant. Data Conversions When mapping columns, you can convert data types, with some restrictions. If you provide '*' or NULL as the column mapping string, only the implicit conversions permitted by SQL are supported. For example, you can convert from CHAR to VARCHAR2, from INTEGER to NUMBER, and so on. If you want to perform other data type conversions, including converting from one object type to another or one collection type to another, you must provide a column mapping pair with an expression that performs the conversion. The expression can include the CAST function, built-in functions like TO_NUMBER, conversion functions that you create, and so on.

12.12.3 Creating Dependent Objects Automatically
You use the COPY_TABLE_DEPENDENTS procedure to automatically create dependent objects on the interim table. You can discover if errors occurred while copying dependent objects by checking the num_errors output argument. If the ignore_errors argument is set to TRUE, the COPY_TABLE_DEPENDENTS procedure continues copying dependent objects even if an error is encountered when creating an object. You can view these errors by querying the DBA_REDEFINITION_ERRORS view.

Reasons for errors include:
A lack of system resources
A change in the logical structure of the table that would require recoding the dependent object. If ignore_errors is set to FALSE, the COPY_TABLE_DEPENDENTS procedure stops copying objects as soon as any error is encountered. After you correct any errors you can again attempt to copy the dependent objects by reexecuting the COPY_TABLE_DEPENDENTS procedure. Optionally you can create the objects manually and then register them. The COPY_TABLE_DEPENDENTS procedure can be used multiple times as necessary. If an object has already been successfully copied, it is not copied again.

12.12.4. Creating Dependent Objects Manually
If you manually create dependent objects on the interim table with SQL*Plus or Enterprise Manager, you must then use the REGISTER_DEPENDENT_OBJECT procedure to register the dependent objects. Registering dependent objects enables the redefinition completion process to restore dependent object names to what they were before redefinition. You would also use the REGISTER_DEPENDENT_OBJECT procedure if the COPY_TABLE_DEPENDENTS procedure failed to copy a dependent object and manual intervention is required. You can query the DBA_REDEFINITION_OBJECTS view to determine which dependent objects are registered. This view shows dependent objects that were registered explicitly with the REGISTER_DEPENDENT_OBJECT procedure or implicitly with the COPY_TABLE_DEPENDENTS procedure. Only current information is shown in the view. The UNREGISTER_DEPENDENT_OBJECT procedure can be used to unregister a dependent object on the table being redefined and on the interim table.

Note: Manually created dependent objects do not have to be identical to their corresponding original dependent objects. For example, when manually creating a materialized view log on the interim table, you can log different columns. In addition, the interim table can have more or fewer dependent objects.

12.12.5. Results of the Redefinition Process
The following are the end results of the redefinition process:
The original table is redefined with the columns, indexes, constraints, grants, triggers, and statistics of the interim table. Dependent objects that were registered, either explicitly using REGISTER_DEPENDENT_OBJECT or implicitly using COPY_TABLE_DEPENDENTS, are renamed automatically so that dependent object names on the redefined table are the same as before redefinition.

Note: If no registration is done or no automatic copying is done, then you must manually rename the dependent objects.

The referential constraints involving the interim table now involve the redefined table and are enabled. Any indexes, triggers, materialized view logs, grants, and constraints defined on the original table (prior to redefinition) are transferred to the interim table and are dropped when the user drops the interim table. Any referential constraints involving the original table before the redefinition now involve the interim table and are disabled. Some PL/SQL
objects that depend on the original table (prior to redefinition) may become invalidated. Only those objects that depend on elements of the table that were changed are invalidated. For example, if a PL/SQL procedure queries only columns of the redefined table that were unchanged by the redefinition, the procedure remains valid.

12.12.6. Performing Intermediate Synchronization

After the redefinition process has been started by calling START_REDEF_TABLE and before FINISH_REDEF_TABLE has been called, it is possible that a large number of DML statements have been executed on the original table. If you know that this is the case, it is recommended that you periodically synchronize the interim table with the original table. This is done by calling the SYNC_INTERIM_TABLE procedure. Calling this procedure reduces the time taken by FINISH_REDEF_TABLE to complete the redefinition process. There is no limit to the number of times that you can call SYNC_INTERIM_TABLE. The small amount of time that the original table is locked during FINISH_REDEF_TABLE is independent of whether SYNC_INTERIM_TABLE has been called.

12.12.7. Aborting Online Table Redefinition and Cleaning Up After Errors

In the event that an error is raised during the redefinition process, or if you choose to terminate the redefinition process, call ABORT_REDEF_TABLE. This procedure drops temporary logs and tables associated with the redefinition process. After this procedure is called, you can drop the interim table and its dependent objects. If the online redefinition process must be restarted, if you do not first call ABORT_REDEF_TABLE, subsequent attempts to redefine the table will fail.

12.12.8. Restrictions for Online Redefinition of Tables

The following restrictions apply to the online redefinition of tables:

- If the table is to be redefined using primary key or pseudo-primary keys (unique keys or constraints with all component columns having not null constraints), then the post-redefinition table must have the same primary key or pseudo-primary key columns. If the table is to be redefined using rowids, then the table must not be an index-organized table.
- After redefining a table that has a materialized view log, the subsequent refresh of any dependent materialized view must be a complete refresh.
- Tables that are replicated in an n-way master configuration can be redefined, but horizontal subsetting (subset of rows in the table), vertical subsetting (subset of columns in the table), and column transformations are not allowed.
- The overflow table of an index-organized table cannot be redefined online independently.
- Tables with fine-grained access control (row-level security) cannot be redefined online.
- Tables with BFILE columns cannot be redefined online.
- Tables with LONG columns can be redefined online, but those columns must be converted to CLOBs. Also, LONG RAW columns must be converted to BLOBs. Tables with LOB columns are acceptable.
- On a system with sufficient resources for parallel execution, and in the case where the interim table is not partitioned, redefinition of a LONG column to a LOB column can be executed in parallel, provided that:
  - The segment used to store the LOB column in the interim table belongs to a locally managed tablespace with Automatic Segment Space Management (ASSM) enabled.
  - There is a simple mapping from one LONG column to one LOB column, and the interim table has only one LOB column.
  - In the case where the interim table is partitioned, the normal methods for parallel execution for partitioning apply.
  - Tables in the SYS and SYSTEM schema cannot be redefined online.
  - Temporary tables cannot be redefined.
  - A subset of rows in the table cannot be redefined.
• Only simple deterministic expressions, sequences, and SYSDATE can be used when mapping the columns in the interim table to those of the original table. For example, subqueries are not allowed.
• If new columns are being added as part of the redefinition and there are no column mappings for these columns, then they must not be declared NOT NULL until the redefinition is complete.
• There cannot be any referential constraints between the table being redefined and the interim table.
• Table redefinition cannot be done NOLOGGING.
• For materialized view logs and queue tables, online redefinition is restricted to changes in physical properties. No horizontal or vertical subsetting is permitted, nor are any column transformations. The only valid value for the column mapping string is NULL.
• You can convert a VARRAY to a nested table with the CAST operator in the column mapping. However, you cannot convert a nested table to a VARRAY.

12.13. Online Table Redefinition Examples

Example 1
This example illustrates online redefinition of the previously created table hr.admin_emp, which at this point only contains columns: empno, ename, job, deptno. The table is redefined as follows:

• New columns mgr, hiredate, sal, and bonus are added. (These existed in the original table but were dropped in previous examples.)
• The new column bonus is initialized to 0
• The column deptno has its value increased by 10.
• The redefined table is partitioned by range on empno.

The steps in this redefinition are illustrated below.
1. Verify that the table is a candidate for online redefinition. In this case you specify that the redefinition is to be done using primary keys or pseudo-primary keys.

BEGIN
  DBMS_REDEFINITION.CAN_REDEF_TABLE('hr', 'admin_emp',
       DBMS_REDEFINITION.CONS_USE_PK);
END;
/

2. Create an interim table hr.int_admin_emp.

CREATE TABLE hr.int_admin_emp
  (empno NUMBER(5) PRIMARY KEY,
   ename VARCHAR2(15) NOT NULL,
   job VARCHAR2(10),
   mgr NUMBER(5),
   hiredate DATE DEFAULT (sysdate),
   sal NUMBER(7,2),
   deptno NUMBER(3) NOT NULL,
   bonus NUMBER (7,2) DEFAULT(1000))
  PARTITION BY RANGE(empno)
    (PARTITION emp1000 VALUES LESS THAN (1000) TABLESPACE admin_tbs,
     PARTITION emp2000 VALUES LESS THAN (2000) TABLESPACE admin_tbs2);

Start the redefinition process.

BEGIN
  DBMS_REDEFINITION.START_REDEF_TABLE('hr', 'admin_emp', 'int_admin_emp',
'empno empno, ename ename, job job, deptno+10 deptno, 0 bonus',
dbms_redefinition.cons_use_pk);
END;
/

4. Copy dependent objects. (Automatically create any triggers, indexes, materialized view logs, grants, and constraints on hr.int_admin_emp.)

DECLARE
num_errors PLS_INTEGER;
BEGIN
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS('hr', 'admin_emp','int_admin_emp',
   DBMS_REDEFINITION.CONS_ORIG_PARAMS, TRUE, TRUE, TRUE, TRUE, num_errors);
END;

Note that the ignore_errors argument is set to TRUE for this call. The reason is that the interim table was created with a primary key constraint, and when COPY_TABLE_DEPENDENTS attempts to copy the primary key constraint and index from the original table, errors occur. You can ignore these errors, but you must run the query shown in the next step to see if there are other errors.

5. Query the DBA_REDEFINITION_ERRORS view to check for errors.

SQL> select object_name, base_table_name, ddl_txt from
   2 DBA_REDEFINITION_ERRORS;

OBJECT_NAME   BASE_TABLE_NAME  DDL_TXT
------------- ---------------- ------------------------------
SYS_C005836   ADMIN_EMP        CREATE UNIQUE INDEX "HR"."TMP$$_SYS_C0058360" ON "HR"."INT_ADMIN_EMP" ("EMPNO")
SYS_C005836   ADMIN_EMP        ALTER TABLE "HR"."INT_ADMIN_EMP" ADD CONSTRAINT "TMP$$_SYS_C0058360" PRIMARY KEY

These errors are caused by the existing primary key constraint on the interim table and can be ignored. Note that with this approach, the names of the primary key constraint and index on the post-redefined table are changed. An alternate approach, one that avoids errors and name changes, would be to define the interim table without a primary key constraint. In this case, the primary key constraint and index are copied from the original table.

Note: The best approach is to define the interim table with a primary key constraint, use REGISTER_DEPENDENT_OBJECT to register the primary key constraint and index, and then copy the remaining dependent objects with COPY_TABLE_DEPENDENTS. This approach avoids errors and ensures that the redefined table always has a primary key and that the dependent object names do not change.

6. Optionally, synchronize the interim table hr.int_admin_emp.

BEGIN
DBMS_REDEFINITION.SYNC_INTERIM_TABLE('hr', 'admin_emp', 'int_admin_emp');
END;
/

7. Complete the redefinition.

BEGIN
DBMS_REDEFINITION.FINISH_REDEF_TABLE('hr', 'admin_emp', 'int_admin_emp');
END;
/
The table hr.admin_emp is locked in the exclusive mode only for a small window toward the end of this step. After this call the table hr.admin_emp is redefined such that it has all the attributes of the hr.int_admin_emp table.

8. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

Example 2
This example redefines a table to change columns into object attributes. The redefined table gets a new column that is an object type. The original table, named CUSTOMER, is defined as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>&lt;- Primary key</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>VARCHAR2 (30)</td>
<td></td>
</tr>
<tr>
<td>STREET</td>
<td>VARCHAR2 (100)</td>
<td></td>
</tr>
<tr>
<td>CITY</td>
<td>VARCHAR2 (30)</td>
<td></td>
</tr>
<tr>
<td>STATE</td>
<td>VARCHAR2 (2)</td>
<td></td>
</tr>
<tr>
<td>ZIP</td>
<td>NUMBER (5)</td>
<td></td>
</tr>
</tbody>
</table>

The type definition for the new object is:

```sql
CREATE TYPE ADDR_T AS OBJECT (street VARCHAR2 (100), city VARCHAR2 (30), state VARCHAR2 (2), zip NUMBER (5,0));
```

Here are the steps for this redefinition:

1. Verify that the table is a candidate for online redefinition. Specify that the redefinition is to be done using primary keys or pseudo-primary keys.

```sql
BEGIN
DBMS_REDEFINITION.CAN_REDEF_TABLE('STEVE','CUSTOMER',
   DBMS_REDEFINITION.CONS_USE_PK);
END;
/
```

2. Create the interim table INT_CUSTOMER.

```sql
CREATE TABLE INT_CUSTOMER(
   CID NUMBER,
   NAME  VARCHAR2 (30),
   ADDR  ADDR_T);
```

Note that no primary key is defined on the interim table. When dependent objects are copied in step 5, the primary key constraint and index are copied.

3. Because CUSTOMER is a very large table, specify parallel operations for the next step.

```sql
alter session force parallel dml parallel 4;
alter session force parallel query parallel 4;
```

4. Start the redefinition process using primary keys.

```sql
BEGIN
DBMS_REDEFINITION.START_REDEF_TABLE(
   uname => 'STEVE',
   orig_table => 'CUSTOMER',
   int_table => 'INT_CUSTOMER',
   col_mapping => 'cid cid, name name,
   addr_t(street, city, state, zip) addr');
```
Note that addr_t(street, city, state, zip) is a call to the object constructor.

5. Copy dependent objects.

```sql
DECLARE
num_errors PLS_INTEGER;
BEGIN
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
   'STEVE', 'CUSTOMER', 'INT_CUSTOMER', DBMS_REDEFINITION.CONS_ORIG_PARAMS,
   TRUE, TRUE, TRUE, FALSE, num_errors, TRUE);
END;
/
```

Note that for this call, the final argument indicates that table statistics are to be copied to the interim table.

6. Optionally synchronize the interim table.

```sql
BEGIN
DBMS_REDEFINITION.SYNC_INTERIM_TABLE('STEVE', 'CUSTOMER', 'INT_CUSTOMER');
END;
/
```

7. Complete the redefinition.

```sql
BEGIN
DBMS_REDEFINITION.FINISH_REDEF_TABLE('STEVE', 'CUSTOMER', 'INT_CUSTOMER');
END;
/
```

8. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

Example 3

This example addresses the situation where a dependent object must be manually created and registered.

Consider the case where a table T1 has a column named C1, and where this column becomes C2 after the redefinition. Assume that there is an index Index1 on C1. In this case, COPY_TABLE_DEPENDENTS tries to create an index on the interim table corresponding to Index1, and tries to create it on a column C1, which does not exist on the interim table. This results in an error. You must therefore manually create the index on column C2 and register it.

Here are the steps:

1. Create the interim table INT_T1 and create an index Int_Index1 on column C2.
2. Ensure that T1 is a candidate for online redefinition with CAN_REDEF_TABLE, and then begin the redefinition process with START_REDEF_TABLE.
3. Register the original (Index1) and interim (Int_Index1) dependent objects.

```sql
BEGIN
DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT(
   uname         => 'STEVE',
   orig_table    => 'T1',
   int_table     => 'INT_T1',
   dep_type      => DBMS_REDEFINITION.CONS_INDEX,
   dep_owner     => 'STEVE',
   dep_orig_name => 'Index1',
   dep_int_name  => 'Int_Index1');
END;
/
```

4. Use COPY_TABLE_DEPENDENTS to copy the remaining dependent objects.
5. Optionally synchronize the interim table.
6. Complete the redefinition and drop the interim table.

**Example 4**
This example demonstrates redefining a single partition. It moves the oldest partition of a range-partitioned sales table to a tablespace named TBS_LOW_FREQ. The table containing the partition to be redefined is defined as follows:

```sql
CREATE TABLE salestable
(s_productid NUMBER,
s_saledate DATE,
s_custid NUMBER,
s_totalprice NUMBER)
TABLESPACE users
PARTITION BY RANGE(s_saledate)
(PARTITION sal03q1 VALUES LESS THAN (TO_DATE('01-APR-2003', 'DD-MON-YYYY')),
PARTITION sal03q2 VALUES LESS THAN (TO_DATE('01-JUL-2003', 'DD-MON-YYYY')),
PARTITION sal03q3 VALUES LESS THAN (TO_DATE('01-OCT-2003', 'DD-MON-YYYY')),
PARTITION sal03q4 VALUES LESS THAN (TO_DATE('01-JAN-2004', 'DD-MON-YYYY')));
```

The table has a local partitioned index that is defined as follows:

```sql
CREATE INDEX sales_index ON salestable (s_saledate, s_productid, s_custid) LOCAL;
```

Here are the steps. In the following procedure calls, note the extra argument: partition name (part_name).

Ensure that salestable is a candidate for redefinition.

```sql
BEGIN
DBMS_REDEFINITION.CAN_REDEF_TABLE(
    uname        => 'STEVE',
    tname        => 'SALESTABLE',
    options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
    part_name    => 'sal03q1');
END;
/
```

2. Create the interim table in the TBS_LOW_FREQ tablespace. Because this is a redefinition of a range partition, the interim table is non-partitioned.

```sql
CREATE TABLE int_salestable
(s_productid NUMBER,
s_saledate DATE,
s_custid NUMBER,
s_totalprice NUMBER)
TABLESPACE tbs_low_freq;
```

3. Start the redefinition process using rowid.

```sql
BEGIN
DBMS_REDEFINITION.START_REDEF_TABLE(
    uname        => 'STEVE',
    orig_table   => 'salestable',
    int_table    => 'int_salestable',
    col_mapping  => NULL,
    options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
    part_name    => 'sal03q1');
END;
```
4. Manually create any local indexes on the interim table.

CREATE INDEX int_sales_index ON int_salesetable (s_saledate, s_productid, s_custid)
TABLESPACE tbs_low_freq;

5. Optionally synchronize the interim table.

BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname => 'STEVE',
    orig_table => 'salestable',
    int_table => 'int_salesetable',
    part_name => 'sal03q1');
END;
/

6. Complete the redefinition.

BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname => 'STEVE',
    orig_table => 'salestable',
    int_table => 'int_salesetable',
    part_name => 'sal03q1');
END;
/

7. Wait for any long-running queries against the interim table to complete, and then drop the interim table. The following query shows that the oldest partition has been moved to the new tablespace:

```
select partition_name, tablespace_name from user_tab_partitions where table_name = 'SALESTABLE';
```

<table>
<thead>
<tr>
<th>PARTITION_NAME</th>
<th>TABLESPACE_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAL03Q1</td>
<td>TBS_LOW_FREQ</td>
</tr>
<tr>
<td>SAL03Q2</td>
<td>USERS</td>
</tr>
<tr>
<td>SAL03Q3</td>
<td>USERS</td>
</tr>
<tr>
<td>SAL03Q4</td>
<td>USERS</td>
</tr>
</tbody>
</table>

4 rows selected.

12.14. Privileges Required for the DBMS_REDEFINITION Package

Execute privileges on the DBMS_REDEFINITION package are granted to EXECUTE_CATALOG_ROLE. In addition to having execute privileges on this package, you must be granted the following privileges:

- CREATE ANY TABLE
- ALTER ANY TABLE
- DROP ANY TABLE
- LOCK ANY TABLE
- SELECT ANY TABLE
The following additional privileges are required to execute COPY_TABLE_DEPENDENTS:

- CREATE ANY TRIGGER
- CREATE ANY INDEX

12.15. Recovering Tables Using Oracle Flashback Table

Oracle Flashback Table enables you to restore a table to its state as of a previous point in time. It provides a fast, online solution for recovering a table that has been accidentally modified or deleted by a user or application. In many cases, Oracle Flashback Table eliminates the need for you to perform more complicated point-in-time recovery operations.

Oracle Flashback Table:

- Restores all data in a specified table to a previous point in time described by a timestamp or SCN.
- Performs the restore operation online.
- Automatically maintains the entire table attributes, such as indexes, triggers, and constraints that are necessary for an application to function with the flashed-back table.
- Maintains any remote state in a distributed environment. For example, all of the table modifications required by replication if a replicated table is flashed back.
- Maintains data integrity as specified by constraints. Tables are flashed back provided none of the table constraints are violated. This includes any referential integrity constraints specified between a table included in the FLASHBACK TABLE statement and another table that is not included in the FLASHBACK TABLE statement.
- Even after a flashback operation, the data in the original table is not lost. You can later revert to the original state.

Note: You must be using automatic undo management to use Oracle Flashback Table.

Dropping Tables

To drop a table that you no longer need, use the DROP TABLE statement. The table must be contained in your schema or you must have the DROP ANY TABLE system privilege.

Caution: Before dropping a table, familiarize yourself with the consequences of doing so:

- Dropping a table removes the table definition from the data dictionary. All rows of the table are no longer accessible.
- All indexes and triggers associated with a table are dropped.
- All views and PL/SQL program units dependent on a dropped table remain, yet become invalid (not usable).
- All synonyms for a dropped table remain, but return an error when used.
- All extents allocated for a table that is dropped are returned to the free space of the tablespace and can be used by any other object requiring new extents or new objects. All rows corresponding to a clustered table are deleted from the blocks of the cluster.

The following statement drops the hr.int_admin_emp table:

```
DROP TABLE hr.int_admin_emp;
```
If the table to be dropped contains any primary or unique keys referenced by foreign keys of other tables and you intend to drop the FOREIGN KEY constraints of the child tables, then include the CASCADE clause in the DROP TABLE statement, as shown below:

```
DROP TABLE hr.admin_emp CASCADE CONSTRAINTS;
```

When you drop a table, normally the database does not immediately release the space associated with the table. Rather, the database renames the table and places it in a recycle bin, where it can later be recovered with the FLASHBACK TABLE statement if you find that you dropped the table in error. If you should want to immediately release the space associated with the table at the time you issue the DROP TABLE statement, include the PURGE clause as shown in the following statement:

```
DROP TABLE hr.admin_emp PURGE;
```

Perhaps instead of dropping a table, you want to truncate it. The TRUNCATE statement provides a fast, efficient method for deleting all rows from a table, but it does not affect any structures associated with the table being truncated (column definitions, constraints, triggers, and so forth) or authorizations.

### 12.15.1. Using Flashback Drop and Managing the Recycle Bin

When you drop a table, the database does not immediately remove the space associated with the table. The database renames the table and places it and any associated objects in a recycle bin, where, in case the table was dropped in error, it can be recovered at a later time. This feature is called Flashback Drop, and the FLASHBACK TABLE statement is used to restore the table. Before discussing the use of the FLASHBACK TABLE statement for this purpose, it is important to understand how the recycle bin works, and how you manage its contents.

### 12.16. What Is the Recycle Bin?

The recycle bin is actually a data dictionary table containing information about dropped objects. Dropped tables and any associated objects such as indexes, constraints, nested tables, and the likes are not removed and still occupy space. They continue to count against user space quotas, until specifically purged from the recycle bin or the unlikely situation where they must be purged by the database because of tablespace space constraints. Each user can be thought of as having his own recycle bin, since unless a user has the SYSDBA privilege, the only objects that the user has access to in the recycle bin are those that the user owns. A user can view his objects in the recycle bin using the following statement:

```
SELECT * FROM RECYCLEBIN;
```

When you drop a tablespace including its contents, the objects in the tablespace are not placed in the recycle bin and the database purges any entries in the recycle bin for objects located in the tablespace. The database also purges any recycle bin entries for objects in a tablespace when you drop the tablespace, not including contents, and the tablespace is otherwise empty. Likewise:

- When you drop a user, any objects belonging to the user are not placed in the recycle bin and any objects in the recycle bin are purged.
- When you drop a cluster, its member tables are not placed in the recycle bin and any former member tables in the recycle bin are purged.
- When you drop a type, any dependent objects such as subtypes are not placed in the recycle bin and any former dependent objects in the recycle bin are purged.

### 12.16.1. Object Naming in the Recycle Bin

When a dropped table is moved to the recycle bin, the table and its associated objects are given system-generated names. This is necessary to avoid name conflicts that may arise if multiple tables have the same name. This could occur under the following circumstances:
• A user drops a table, re-creates it with the same name, then drops it again.
• Two users have tables with the same name, and both users drop their tables.

The renaming convention is as follows:

```
BIN$unique_id$version
```

Where:
- unique_id is a 26-character globally unique identifier for this object, which makes the recycle bin name unique across all databases.
- version is a version number assigned by the database.

12.16.2. Enabling and Disabling the Recycle Bin

You can enable and disable the recycle bin with the recyclebin initialization parameter. When the recycle bin is enabled, dropped tables and their dependent objects are placed in the recycle bin. When the recycle bin is disabled, dropped tables and their dependent objects are not placed in the recycle bin; they are just dropped, and you must use other means to recover them (such as recovering from backup). The recycle bin is enabled by default.

To disable the recycle bin: Issue one of the following statements:

```
ALTER SESSION SET recyclebin = OFF;
ALTER SYSTEM SET recyclebin = OFF;
```

To enable the recycle bin: Issue one of the following statements:

```
ALTER SESSION SET recyclebin = ON;
ALTER SYSTEM SET recyclebin = ON;
```

Enabling and disabling the recycle bin with an ALTER SYSTEM or ALTER SESSION statement takes effect immediately. Disabling the recycle bin does not purge or otherwise affect objects already in the recycle bin. Like any other initialization parameter, you can set the initial value of the recyclebin parameter in the text initialization file initSID.ora:

```
recyclebin=on
```

12.16.3. Viewing and Querying Objects in the Recycle Bin

Oracle Database provides two views for obtaining information about objects in the recycle bin:

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_RECYCLEBIN</td>
<td>This view can be used by users to see their own dropped objects in the recycle bin. It has a synonym RECYCLEBIN, for ease of use.</td>
</tr>
<tr>
<td>DBA_RECYCLEBIN</td>
<td>This view gives administrators visibility to all dropped objects in the recycle bin</td>
</tr>
</tbody>
</table>

One use for these views is to identify the name that the database has assigned to a dropped object, as shown in the following example:

```
SELECT object_name, original_name FROM dba_reyclebin WHERE owner = 'HR';
```

```
object_name          original_name
----------------------- -----------------------
BIN$yrMK1zaLMhfgNAgAImenRA==0$ EMPLOYEES
```

You can also view the contents of the recycle bin using the SQL*Plus command SHOW RECYCLEBIN.
You can query objects that are in the recycle bin, just as you can query other objects. However, you must specify the name of the object as it is identified in the recycle bin.

For example:

```
SELECT * FROM "BIN$yrMKlZaVMhfNAGA1MenRA==$0";
```

### 12.16.4. Purging Objects in the Recycle Bin

If you decide that you are never going to restore an item from the recycle bin, you can use the PURGE statement to remove the items and their associated objects from the recycle bin and release their storage space. You need the same privileges as if you were dropping the item. When you use the PURGE statement to purge a table, you can use the name that the table is known by in the recycle bin or the original name of the table. The following hypothetical example purges the table `hr.int_admin_emp`, which was renamed to `BIN$jsleilx392mk2=293$0` when it was placed in the recycle bin:

```
PURGE TABLE BIN$jsleilx392mk2=293$0;
```

You can achieve the same result with the following statement:

```
PURGE TABLE int_admin_emp;
```

You can use the PURGE statement to purge all the objects in the recycle bin that are from a specified tablespace or only the tablespace objects belonging to a specified user, as shown in the following examples:

```
PURGE TABLESPACE example;
PURGE TABLESPACE example USER oe;
```

Users can purge the recycle bin of their own objects, and release space for objects, by using the following statement:

```
PURGE RECYCLEBIN;
```

If you have the SYSDBA privilege, then you can purge the entire recycle bin by specifying `DBA_RECYCLEBIN`, instead of `RECYCLEBIN` in the previous statement. You can also use the PURGE statement to purge an index from the recycle bin or to purge from the recycle bin all objects in a specified tablespace.

### 12.16.5. Restoring Tables from the Recycle Bin

Use the `FLASHBACK TABLE ... TO BEFORE DROP` statement to recover objects from the recycle bin. You can specify either the name of the table in the recycle bin or the original table name. An optional `RENAME TO` clause lets you rename the table as you recover it. The recycle bin name can be obtained from either the `DBA_` or `USER_RECYCLEBIN` view. To use the `FLASHBACK TABLE ... TO BEFORE DROP` statement, you need the same privileges you need to drop the table. The following example restores `int_admin_emp` table and assigns to it a new name:

```
FLASHBACK TABLE int_admin_emp TO BEFORE DROP RENAME TO int2_admin_emp;
```

The system-generated recycle bin name is very useful if you have dropped a table multiple times. For example, suppose you have three versions of the `int2_admin_emp` table in the recycle bin and you want to recover the second version. You can do this by issuing two `FLASHBACK TABLE` statements, or you can query the recycle bin and then flashback to the appropriate system-generated name, as shown in the following example. Including the create time in the query can help you verify that you are restoring the correct table.
SELECT object_name, original_name, createtime FROM recyclebin;

<table>
<thead>
<tr>
<th>OBJECT_NAME</th>
<th>ORIGINAL_NAME</th>
<th>CREATETIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIN$yrMKlZaLMhfgNaAGMenRA==$0</td>
<td>INT2_ADMIN_EMP</td>
<td>2006-02-05:21:05:52</td>
</tr>
<tr>
<td>BIN$yrMKlZaVMhfgNaAGMenRA==$0</td>
<td>INT2_ADMIN_EMP</td>
<td>2006-02-05:21:25:13</td>
</tr>
<tr>
<td>BIN$yrMKlZaQMhfgNaAGMenRA==$0</td>
<td>INT2_ADMIN_EMP</td>
<td>2006-02-05:22:05:53</td>
</tr>
</tbody>
</table>

FLASHBACK TABLE BIN$yrMKlZaVMhfgNaAGMenRA==$0 TO BEFORE DROP;

12.16.6. Restoring Dependent Objects

When you restore a table from the recycle bin, dependent objects such as indexes do not get their original names back; they retain their system-generated recycle bin names. You must manually rename dependent objects if you want to restore their original names. If you plan to manually restore original names for dependent objects, ensure that you make note of each dependent object's system-generated recycle bin name before you restore the table. The following is an example of restoring the original names of some of the indexes of the dropped table JOB_HISTORY, from the HR sample schema. The example assumes that you are logged in as the HR user.

1. After dropping JOB_HISTORY and before restoring it from the recycle bin, run the following query:

SELECT OBJECT_NAME, ORIGINAL_NAME, TYPE FROM RECYCLEBIN;

<table>
<thead>
<tr>
<th>OBJECT_NAME</th>
<th>ORIGINAL_NAME</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIN$DBo9UChtZSbgQFeMiAdCcQ==$0</td>
<td>JHIST_JOB_IX</td>
<td>INDEX</td>
</tr>
<tr>
<td>BIN$DBo9UChuZSbgQFeMiAdCcQ==$0</td>
<td>JHIST_EMPLOYEE_IX</td>
<td>INDEX</td>
</tr>
<tr>
<td>BIN$DBo9UChvZSbgQFeMiAdCcQ==$0</td>
<td>JHIST_DEPARTMENT_IX</td>
<td>INDEX</td>
</tr>
<tr>
<td>BIN$DBo9UChwZSbgQFeMiAdCcQ==$0</td>
<td>JHIST_EMP_ID_ST_DATE_PK</td>
<td>INDEX</td>
</tr>
<tr>
<td>BIN$DBo9UChxZSbgQFeMiAdCcQ==$0</td>
<td>JOB_HISTORY</td>
<td>TABLE</td>
</tr>
</tbody>
</table>

2. Restore the table with the following command:

FLASHBACK TABLE JOB_HISTORY TO BEFORE DROP;

3. Run the following query to verify that all JOB_HISTORY indexes retained their system-generated recycle bin names:

SELECT INDEX_NAME FROM USER_INDEXES WHERE TABLE_NAME = 'JOB_HISTORY';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIN$DBo9UChwZSbgQFeMiAdCcQ==0</td>
</tr>
<tr>
<td>BIN$DBo9UChtZSbgQFeMiAdCcQ==0</td>
</tr>
<tr>
<td>BIN$DBo9UChvZSbgQFeMiAdCcQ==0</td>
</tr>
<tr>
<td>BIN$DBo9UChxZSbgQFeMiAdCcQ==0</td>
</tr>
</tbody>
</table>

4. Restore the original names of the first two indexes as follows:

ALTER INDEX "BIN$DBo9UChwZSbgQFeMiAdCcQ==0" RENAME TO JHIST_JOB_IX;
ALTER INDEX "BIN$DBo9UChtZSbgQFeMiAdCcQ==0" RENAME TO JHIST_EMPLOYEE_IX;

Note that double quotes are required around the system-generated names.
12.17. Managing Index-Organized Tables

12.17.1. What Are Index-Organized Tables?
An index-organized table has a storage organization that is a variant of a primary B-tree. Unlike an ordinary (heap-organized) table whose data is stored as an unordered collection (heap), data for an index-organized table is stored in a B-tree index structure in a primary key sorted manner. Each leaf block in the index structure stores both the key and non-key columns. The structure of an index-organized table provides the following benefits:

- Fast random access on the primary key because an index-only scan is sufficient. And, because there is no separate table storage area, changes to the table data (such as adding new rows, updating rows, or deleting rows) result only in updating the index structure.
- Fast range access on the primary key because the rows are clustered in primary key order.

Lower storage requirements because duplication of primary keys is avoided. They are not stored both in the index and underlying table, as is true with heap-organized tables. Index-organized tables have full table functionality. They support features such as constraints, triggers, LOB and object columns, partitioning, parallel operations, online reorganization, and replication. And, they offer these additional features:

Index-organized tables are ideal for OLTP applications, which require fast primary key access and high availability. Queries and DML on an orders table used in electronic order processing are predominantly primary-key based and heavy volume causes fragmentation resulting in a frequent need to reorganize. Because an index-organized table can be reorganized online and without invalidating its secondary indexes, the window of unavailability is greatly reduced or eliminated. Index-organized tables are suitable for modeling application-specific index structures. For example, content-based information retrieval applications containing text, image and audio data require inverted indexes that can be effectively modeled using index-organized tables. A fundamental component of an internet search engine is an inverted index that can be modeled using index-organized tables. These are but a few of the applications for index-organized tables.

12.17.2. Creating Index-Organized Tables
You use the CREATE TABLE statement to create index-organized tables, but you must provide additional information:

An ORGANIZATION INDEX qualifier, which indicates that this is an index-organized table
A primary key, specified through a column constraint clause (for a single column primary key) or a table constraint clause (for a multiple-column primary key). Optionally, you can specify the following:

- An OVERFLOW clause, which preserves dense clustering of the B-tree index by enabling the storage of some of the non-key columns in a separate overflow data segment.
- A PCTTHRESHOLD value, which, when an overflow segment is being used, defines the maximum size of the portion of the row that is stored in the index block, as a percentage of block size. Rows columns that would cause the row size to exceed this maximum are stored in the overflow segment. The row is broken at a column boundary into two pieces, a head piece and tail piece. The head piece fits in the specified threshold and is stored along with the key in the index leaf block. The tail piece is stored in the overflow area as one or more row pieces. Thus, the index entry contains the key value, the non-key column values that fit the specified threshold, and a pointer to the rest of the row.
- An INCLUDING clause, which can be used to specify the non-key columns that are to be stored in the index block with the primary key.

Example: Creating an Index-Organized Table
The following statement creates an index-organized table:
CREATE TABLE admin_docindex(
    token char(20),
    doc_id NUMBER,
    token_frequency NUMBER,
    token_offsets VARCHAR2(2000),
    CONSTRAINT pk_admin_docindex PRIMARY KEY (token, doc_id)
) ORGANIZATION INDEX
TABLESPACE admin_tbs
PCTTHRESHOLD 20
OVERFLOW TABLESPACE admin_tbs2;

This example creates an index-organized table named admin_docindex, with a primary key composed of the columns token and doc_id. The OVERFLOW and PCTTHRESHOLD clauses specify that if the length of a row exceeds 20% of the index block size, then the column that exceeded that threshold and all columns after it are moved to the overflow segment. The overflow segment is stored in the admin_tbs2 tablespace.

12.17.3. Restrictions for Index-Organized Tables

- The following are restrictions on creating index-organized tables.
- The maximum number of columns is 1000.
- The maximum number of columns in the index portion of a row is 255, including both key and non-key columns. If more than 255 columns are required, you must use an overflow segment.
- The maximum number of columns that you can include in the primary key is 32.
- PCTTHRESHOLD must be in the range of 1–50. The default is 50.
- All key columns must fit within the specified threshold.
- If the maximum size of a row exceeds 50% of the index block size and you do not specify an overflow segment, the CREATE TABLE statement fails.
- Index-organized tables cannot have virtual columns.

Creating Index-Organized Tables that Contain Object Types

Index-organized tables can store object types. The following example creates object type admin_typ, then creates an index-organized table containing a column of object type admin_typ:

CREATE OR REPLACE TYPE admin_typ AS OBJECT (col1 NUMBER, col2 VARCHAR2(6));
CREATE TABLE admin_iot (c1 NUMBER primary key, c2 admin_typ) ORGANIZATION INDEX;

You can also create an index-organized table of object types. For example:

CREATE TABLE admin_iot2 OF admin_typ (col1 PRIMARY KEY) ORGANIZATION INDEX;

Another example, that follows, shows that index-organized tables store nested tables efficiently. For a nested table column, the database internally creates a storage table to hold all the nested table rows.

CREATE TYPE project_t AS OBJECT(pno NUMBER, pname VARCHAR2(80));
/CREATE TYPE project_set AS TABLE OF project_t;
/
CREATE TABLE proj_tab (eno NUMBER, projects PROJECT_SET) NESTED TABLE projects STORE AS emp_project_tab ((PRIMARY KEY (nested_table_id, pno)) ORGANIZATION INDEX) RETURN AS LOCATOR;

The rows belonging to a single nested table instance are identified by a nested_table_id column. If an ordinary table
is used to store nested table columns, the nested table rows typically get de-clustered. But when you use an index-organized table, the nested table rows can be clustered based on the nested_table_id column.

12.17.4. Choosing and Monitoring a Threshold Value
Choose a threshold value that can accommodate your key columns, as well as the first few non-key columns (if they are frequently accessed). After choosing a threshold value, you can monitor tables to verify that the value you specified is appropriate. You can use the ANALYZE TABLE ... LIST CHAINED ROWS statement to determine the number and identity of rows exceeding the threshold value.

12.17.5. Using the INCLUDING Clause
In addition to specifying PCTTHRESHOLD, you can use the INCLUDING clause to control which non-key columns are stored with the key columns. The database accommodates all non-key columns up to and including the column specified in the INCLUDING clause in the index leaf block, provided it does not exceed the specified threshold. All non-key columns beyond the column specified in the INCLUDING clause are stored in the overflow segment. If the INCLUDING and PCTTHRESHOLD clauses conflict, PCTTHRESHOLD takes precedence.

Note: Oracle Database moves all primary key columns of an indexed-organized table to the beginning of the table (in their key order) to provide efficient primary key–based access. As an example:

```
CREATE TABLE admin_iot4(a INT, b INT, c INT, d INT, primary key(c,b))
ORGANIZATION INDEX;
```

The stored column order is: c b a d (instead of: a b c d). The last primary key column is b, based on the stored column order. The INCLUDING column can be the last primary key column (b in this example), or any nonkey column (that is, any column after b in the stored column order).

The following CREATE TABLE statement is similar to the one shown earlier but is modified to create an index-organized table where the token_offsets column value is always stored in the overflow area:

```
CREATE TABLE admin_docindex2(
  token CHAR(20),
  doc_id NUMBER,
  token_frequency NUMBER,
  token_offsets VARCHAR2(2000),
  CONSTRAINT pk_admin_docindex2 PRIMARY KEY (token, doc_id))
ORGANIZATION INDEX
TABLESPACE admin_tbs
PCTTHRESHOLD 20
INCLUDING token_frequency
OVERFLOW TABLESPACE admin_tbs2;
```

Here, only nonkey columns prior to token_offsets (in this case a single column only) are stored with the key column values in the index leaf block.

12.17.6. Parallelizing Index-Organized Table Creation
The CREATE TABLE...AS SELECT statement enables you to create an index-organized table and load data from an existing table into it. By including the PARALLEL clause, the load can be done in parallel. The following statement creates an index-organized table in parallel by selecting rows from the conventional table hr.jobs:

```
CREATE TABLE admin_iot3(i PRIMARY KEY, j, k, l) ORGANIZATION INDEX PARALLEL AS
SELECT * FROM hr.jobs;
```

This statement provides an alternative to parallel bulk-load using SQL*Loader.
12.17.7. Using Key Compression

Creating an index-organized table using key compression enables you to eliminate repeated occurrences of key column prefix values. Key compression breaks an index key into a prefix and a suffix entry. Compression is achieved by sharing the prefix entries among all the suffix entries in an index block. This sharing can lead to huge savings in space, allowing you to store more keys in each index block while improving performance. You can enable key compression using the COMPRESS clause while:

You can also specify the prefix length (as the number of key columns), which identifies how the key columns are broken into a prefix and suffix entry.

```
CREATE TABLE admin_iot5(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
    ORGANIZATION INDEX COMPRESS;
```

The preceding statement is equivalent to the following statement:

```
CREATE TABLE admin_iot6(i INT, j INT, k INT, l INT, PRIMARY KEY(i, j, k))
    ORGANIZATION INDEX COMPRESS 2;
```

For the list of values (1,2,3), (1,2,4), (1,2,7), (1,3,5), (1,3,4), (1,4,4) the repeated occurrences of (1,2), (1,3) are compressed away. You can also override the default prefix length used for compression as follows:

```
CREATE TABLE admin_iot7(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
    ORGANIZATION INDEX COMPRESS 1;
```

For the list of values (1,2,3), (1,2,4), (1,2,7), (1,3,5), (1,3,4), (1,4,4), the repeated occurrences of 1 are compressed away. You can disable compression as follows:

```
ALTER TABLE admin_iot5 MOVE NOCOMPRESS;
```

One application of key compression is in a time-series application that uses a set of time-stamped rows belonging to a single item, such as a stock price. Index-organized tables are attractive for such applications because of the ability to cluster rows based on the primary key. By defining an index-organized table with primary key (stock symbol, time stamp), you can store and manipulate time-series data efficiently. You can achieve more storage savings by compressing repeated occurrences of the item identifier (for example, the stock symbol) in a time series by using an index-organized table with key compression.

12.17.8. Maintaining Index-Organized Tables

Index-organized tables differ from ordinary tables only in physical organization. Logically, they are manipulated in the same manner as ordinary tables. You can specify an index-organized table just as you would specify a regular table in INSERT, SELECT, DELETE, and UPDATE statements.

12.17.9. Altering Index-Organized Tables

All of the alter options available for ordinary tables are available for index-organized tables. This includes ADD, MODIFY, and DROP COLUMNS and CONSTRAINTS. However, the primary key constraint for an index-organized table cannot be dropped, deferred, or disabled

You can use the ALTER TABLE statement to modify physical and storage attributes for both primary key index and overflow data segments. All the attributes specified prior to the OVERFLOW keyword are applicable to the primary key index segment. All attributes specified after the OVERFLOW key word are applicable to the overflow data segment. For example, you can set the INITRANS of the primary key index segment to 4 and the overflow of the data segment INITRANS to 6 as follows:

```
ALTER TABLE admin_docindex INITRANS 4 OVERFLOW INITRANS 6;
```

You can also alter PCTTHRESHOLD and INCLUDING column values. A new setting is used to break the row into head and overflow tail pieces during subsequent operations. For example, the PCTHRESHOLD and INCLUDING column values can be altered for the admin_docindex table as follows:
ALTER TABLE admin_docindex PCTTHRESHOLD 15 INCLUDING doc_id;

By setting the INCLUDING column to doc_id, all the columns that follow token_frequency and token_offets, are stored in the overflow data segment. For index-organized tables created without an overflow data segment, you can add an overflow data segment by using the ADD OVERFLOW clause. For example, you can add an overflow segment to table admin_iot3 as follows:

ALTER TABLE admin_iot3 ADD OVERFLOW TABLESPACE admin_tbs2;

12.17.10. Moving (Rebuilding) Index-Organized Tables

Because index-organized tables are primarily stored in a B-tree index, you can encounter fragmentation as a consequence of incremental updates. However, you can use the ALTER TABLE...MOVE statement to rebuild the index and reduce this fragmentation. The following statement rebuilds the index-organized table admin_docindex:

ALTER TABLE admin_docindex MOVE;

You can rebuild index-organized tables online using the ONLINE keyword. The overflow data segment, if present, is rebuilt when the OVERFLOW keyword is specified. For example, to rebuild the admin_docindex table but not the overflow data segment, perform a move online as follows:

ALTER TABLE admin_docindex MOVE ONLINE;

To rebuild the admin_docindex table along with its overflow data segment perform the move operation as shown in the following statement. This statement also illustrates moving both the table and overflow data segment to new tablespaces.

ALTER TABLE admin_docindex MOVE TABLESPACE admin_tbs2
  OVERFLOW TABLESPACE admin_tbs3;

In this last statement, an index-organized table with a LOB column (CLOB) is created. Later, the table is moved with the LOB index and data segment being rebuilt and moved to a new tablespace.

CREATE TABLE admin_iot_lob
  (c1 number (6) primary key,
   admin_lob CLOB)
  ORGANIZATION INDEX
  LOB (admin_lob) STORE AS (TABLESPACE admin_tbs2);

ALTER TABLE admin_iot_lob MOVE LOB (admin_lob) STORE AS (TABLESPACE admin_tbs3);

12.17.11. Creating Secondary Indexes on Index-Organized Tables

You can create secondary indexes on index-organized tables to provide multiple access paths. Secondary indexes on index-organized tables differ from indexes on ordinary tables in two ways:

They store logical rowids instead of physical rowids. This is necessary because the inherent movability of rows in a B-tree index results in the rows having no permanent physical addresses. If the physical location of a row changes, its logical rowid remains valid. One effect of this is that a table maintenance operation, such as ALTER TABLE ... MOVE, does not make the secondary index unusable. The logical rowid also includes a physical guess which identifies the database block address at which the row is likely to be found. If the physical guess is correct, a secondary index scan would incur a single additional I/O once the secondary key is found. The performance would be similar to that of a secondary index-scan on an ordinary table. Unique and non-unique secondary indexes, function-based secondary indexes, and bitmap indexes are supported as secondary indexes on index-organized
12.17.12. Creating a Secondary Index on an Index-Organized Table

The following statement shows the creation of a secondary index on the docindex index-organized table where doc_id and token are the key columns:

```
CREATE INDEX Doc_id_index on Docindex(Doc_id, Token);
```

This secondary index allows the database to efficiently process a query, such as the following, involves a predicate on doc_id:

```
SELECT Token FROM Docindex WHERE Doc_id = 1;
```

Use the ANALYZE statement if you want to validate the structure of your index-organized table or to list any chained rows.

**Note:** There are special considerations when listing chained rows for index-organized tables.

12.17.13. Using the ORDER BY Clause with Index-Organized Tables

If an ORDER BY clause only references the primary key column or a prefix of it, then the optimizer avoids the sorting overhead, as the rows are returned sorted on the primary key columns. The following queries avoid sorting overhead because the data is already sorted on the primary key:

```
SELECT * FROM admin_docindex2 ORDER BY token, doc_id;
SELECT * FROM admin_docindex2 ORDER BY token;
```

If, however, you have an ORDER BY clause on a suffix of the primary key column or non-primary-key columns, additional sorting is required (assuming no other secondary indexes are defined).

```
SELECT * FROM admin_docindex2 ORDER BY doc_id;
SELECT * FROM admin_docindex2 ORDER BY token_frequency;
```

12.17.14. Converting Index-Organized Tables to Regular Tables

You can convert index-organized tables to regular tables using the Oracle import or export utilities, or the CREATE TABLE...AS SELECT statement. To convert an index-organized table to a regular table:

- Export the index-organized table data using conventional path.
- Create a regular table definition with the same definition.
- Import the index-organized table data, making sure IGNORE=y (ensures that object exists error is ignored).

**Note:** Before converting an index-organized table to a regular table, be aware that index-organized tables cannot be exported using pre-Oracle8 versions of the Export utility.

12.18. Managing External Tables

Oracle Database allows you read-only access to data in external tables. External tables are defined as tables that do not reside in the database, and can be in any format for which an access driver is provided. By providing the database with metadata describing an external table, the database is able to expose the data in the external table as if it were data residing in a regular database table. The external data can be queried directly and in parallel using SQL.

You can, for example, select, join, or sort external table data. You can also create views and synonyms for external
tables. However, no DML operations (UPDATE, INSERT, or DELETE) are possible, and no indexes can be created, on external tables. External tables also provide a framework to unload the result of an arbitrary SELECT statement into a platform-independent Oracle-proprietary format that can be used by Oracle Data Pump.

**Note:** The DBMS_STATS package can be used for gathering statistics for external tables. The ANALYZE statement is not supported for gathering statistics for external tables.

The means of defining the metadata for external tables is through the CREATE TABLE...ORGANIZATION EXTERNAL statement. This external table definition can be thought of as a view that allows running any SQL query against external data without requiring that the external data first be loaded into the database. An access driver is the actual mechanism used to read the external data in the table. When you use external tables to unload data, the metadata is automatically created based on the datatypes in the SELECT statement (sometimes referred to as the shape of the query).

Oracle Database provides two access drivers for external tables. The default access driver is ORACLE_LOADER, which allows the reading of data from external files using the Oracle loader technology. The ORACLE_LOADER access driver provides data mapping capabilities which are a subset of the control file syntax of SQL*Loader utility. The second access driver, ORACLE_DATAPUMP, lets you unload data—that is, read data from the database and insert it into an external table, represented by one or more external files—and then reload it into an Oracle Database.

The Oracle Database external tables feature provides a valuable means for performing basic extraction, transformation, and loading (ETL) tasks that are common for data warehousing. These following sections discuss the DDL statements that are supported for external tables. Only DDL statements discussed are supported, and not all clauses of these statements are supported.

### 12.18.1. Creating External Tables

You create external tables using the ORGANIZATION EXTERNAL clause of the CREATE TABLE statement. You are not in fact creating a table; that is, an external table does not have any extents associated with it. Rather, you are creating metadata in the data dictionary that enables you to access external data.

**Note:** External tables cannot have virtual columns.

The following example creates an external table and then uploads the data to a database table. Alternatively, you can unload data through the external table framework by specifying the AS subquery clause of the CREATE TABLE statement. External table data pump unload can use only the ORACLE_DATAPUMP access driver.

**EXAMPLE:** Creating an External Table and Loading Data

The file empxt1.dat contains the following sample data:

```
360, Jane, Janus, ST_CLERK, 121, 17-MAY-2001, 3000, 0, 50, jjanus
361, Mark, Jasper, SA_REP, 145, 17-MAY-2001, 8000, .1, 80, mjasper
362, Brenda, Starr, AD_ASST, 200, 17-MAY-2001, 5500, 0, 10, bstarr
363, Alex, Aida, AC_MGR, 145, 17-MAY-2001, 9000, .15, 80, aaida
```

The file empxt2.dat contains the following sample data:

```
401, Jesse, Cromwell, HR_REP, 203, 17-MAY-2001, 7000, 0, 40, jcromwel
402, Abby, Applegate, IT_PROG, 103, 17-MAY-2001, 9000, .2, 60, aapplega
403, Carol, Cousins, AD_VP, 100, 17-MAY-2001, 27000, .3, 90, ccousins
404, John, Richardson, AC_ACCOUNT, 205, 17-MAY-2001, 5000, 0, 110, jrichard
```

The following hypothetical SQL statements create an external table in the hr schema named admin_ext_employees and load its data into the hr.employees table.

```
CONNECT / AS SYSDBA;
```
-- Set up directories and grant access to hr

CREATE OR REPLACE DIRECTORY admin_dat_dir
AS '/flatfiles/data';
CREATE OR REPLACE DIRECTORY admin_log_dir
AS '/flatfiles/log';
CREATE OR REPLACE DIRECTORY admin_bad_dir
AS '/flatfiles/bad';
GRANT READ ON DIRECTORY admin_dat_dir TO hr;
GRANT WRITE ON DIRECTORY admin_log_dir TO hr;
GRANT WRITE ON DIRECTORY admin_bad_dir TO hr;

-- hr connects. Provide the user password (hr) when prompted.

CONNECT hr

-- create the external table

CREATE TABLE admin_ext_employees
(employee_id NUMBER(4),
 first_name VARCHAR2(20),
 last_name VARCHAR2(25),
 job_id VARCHAR2(10),
 manager_id NUMBER(4),
 hire_date DATE,
 salary NUMBER(8,2),
 commission_pct NUMBER(2,2),
 department_id NUMBER(4),
 email VARCHAR2(25)
 )

ORGANIZATION EXTERNAL
(
 TYPE ORACLE_LOADER
 DEFAULT DIRECTORY admin_dat_dir
 ACCESS PARAMETERS
 ( records delimited by newline
 badfile admin_bad_dir:'empxt%a_%p.bad'
 logfile admin_log_dir:'empxt%a_%p.log'
 fields terminated by ','
 missing field values are null
 ( employee_id, first_name, last_name, job_id, manager_id,
   hire_date char date_format date mask "dd-mon-yyyy",
   salary, commission_pct, department_id, email
 )
)
 LOCATION ('empxt1.dat', 'empxt2.dat')
)

PARALLEL
REJECT LIMIT UNLIMITED;

-- enable parallel for loading (good if lots of data to load)

ALTER SESSION ENABLE PARALLEL DML;

-- load the data in hr employees table
INSERT INTO employees (employee_id, first_name, last_name, job_id, manager_id, hire_date, salary, commission_pct, department_id, email)
    SELECT * FROM admin_ext_employees;

The following paragraphs contain descriptive information about this example. The first few statements in this example create the directory objects for the operating system directories that contain the data sources, and for the bad record and log files specified in the access parameters. You must also grant READ or WRITE directory object privileges, as appropriate.

**Note:** When creating a directory object or BFILEs, ensure that the following conditions are met:

- The operating system file must not be a symbolic or hard link.
- The operating system directory path named in the Oracle Database directory object must be an existing OS directory path.
- The operating system directory path named in the directory object should not contain any symbolic links in its components.

The TYPE specification indicates the access driver of the external table. The access driver is the API that interprets the external data for the database. Oracle Database provides two access drivers: ORACLE_LOADER and ORACLE_DATAPUMP. If you omit the TYPE specification, ORACLE_LOADER is the default access driver. You must specify the ORACLE_DATAPUMP access driver if you specify the AS subquery clause to unload data from one Oracle Database and reload it into the same or a different Oracle Database.

The access parameters, specified in the ACCESS PARAMETERS clause, are opaque to the database. These access parameters are defined by the access driver, and are provided to the access driver by the database when the external table is accessed. The PARALLEL clause enables parallel query on the data sources. The granule of parallelism is by default a data source, but parallel access within a data source is implemented whenever possible. For example, if PARALLEL=3 were specified, then more than one parallel execution server could be working on a data source. But, parallel access within a data source is provided by the access driver only if all of the following conditions are met:

- The media allows random positioning within a data source
- It is possible to find a record boundary from a random position
- The data files are large enough to make it worthwhile to break up into multiple chunks

**Note:** Specifying a PARALLEL clause is of value only when dealing with large amounts of data. Otherwise, it is not advisable to specify a PARALLEL clause, and doing so can be detrimental.

The REJECT LIMIT clause specifies that there is no limit on the number of errors that can occur during a query of the external data. For parallel access, this limit applies to each parallel execution server independently. For example, if REJECT LIMIT is specified, each parallel query process is allowed 10 rejections. Hence, the only precisely enforced values for REJECT LIMIT on parallel query are 0 and UNLIMITED. In this example, the INSERT INTO TABLE statement generates a dataflow from the external data source to the Oracle Database SQL engine where data is processed. As data is parsed by the access driver from the external table sources and provided to the external table interface, the external data is converted from its external representation to its Oracle Database internal datatype.

### 12.18.2. Altering External Tables

You can use any of the ALTER TABLE clauses shown in Table to change the characteristics of an external table. No other clauses are permitted.

### 12.18.3. Dropping External Tables

For an external table, the DROP TABLE statement removes only the table metadata in the database. It has no affect
on the actual data, which resides outside of the database.

12.18.4. System and Object Privileges for External Tables

System and object privileges for external tables are a subset of those for regular table. Only the following system privileges are applicable to external tables:

- CREATE ANY TABLE
- ALTER ANY TABLE
- DROP ANY TABLE
- SELECT ANY TABLE

Only the following object privileges are applicable to external tables:

- ALTER
- SELECT

However, object privileges associated with a directory are:

- READ
- WRITE

For external tables, READ privileges are required on directory objects that contain data sources, while WRITE privileges are required for directory objects containing bad, log, or discard files.
12.18.5. Tables Data Dictionary Views

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA_TABLES</td>
<td>DBA view describes all relational tables in the database. ALL view describes all tables accessible to the user. USER view is restricted to tables owned by the user. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.</td>
</tr>
<tr>
<td>ALL_TABLES</td>
<td>These views describe all tables in the database. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.</td>
</tr>
<tr>
<td>USER_TABLES</td>
<td>These views describe all relational and object tables in the database. Object tables are not specifically discussed in this book.</td>
</tr>
<tr>
<td>DBA_TAB_COLUMNS</td>
<td>These views display comments for tables and views. Comments are entered using the COMMENT statement.</td>
</tr>
<tr>
<td>ALL_TAB_COLUMNS</td>
<td>These views display comments for table and view columns. Comments are entered using the COMMENT statement.</td>
</tr>
<tr>
<td>USER_TAB_COLUMNS</td>
<td>These views list the specific attributes of external tables in the database.</td>
</tr>
<tr>
<td>DBA_EXTERNAL_TABLES</td>
<td>These views list the data sources for external tables.</td>
</tr>
<tr>
<td>ALL_EXTERNAL_TABLES</td>
<td>These views describe histograms on tables and views.</td>
</tr>
<tr>
<td>USER_EXTERNAL_TABLES</td>
<td>These views contain optimizer statistics for tables.</td>
</tr>
<tr>
<td>DBA_TAB_STATISTICS</td>
<td>These views provide column statistics and histogram information extracted from the related TAB_COLUMNS views.</td>
</tr>
<tr>
<td>ALL_TAB_STATISTICS</td>
<td>These views describe tables that have been modified since the last time table statistics were gathered on them. They are not populated immediately, but after a time lapse (usually 3 hours).</td>
</tr>
<tr>
<td>USER_TAB_STATISTICS</td>
<td></td>
</tr>
</tbody>
</table>

The following views allow you to access information about tables.

**Example: Displaying Column Information**

Column information, such as name, datatype, length, precision, scale, and default data values can be listed using one of the views ending with the _COLUMNS suffix. For example, the following query lists all of the default column values for the emp and dept tables:

```sql
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGTH, LAST_ANALYZED FROM DBA_TAB_COLUMNS WHERE OWNER = 'HR' ORDER BY TABLE_NAME;
```
The following is the output from the query:

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>DATA_TYPE</th>
<th>DATA_LENGTH</th>
<th>LAST_ANALYZED</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRIES</td>
<td>COUNTRY_ID</td>
<td>CHAR</td>
<td>2</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>COUNTRIES</td>
<td>COUNTRY_NAME</td>
<td>VARCHAR2</td>
<td>40</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>COUNTRIES</td>
<td>REGION_ID</td>
<td>NUMBER</td>
<td>22</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>DEPARTMENTS</td>
<td>DEPARTMENT_ID</td>
<td>NUMBER</td>
<td>22</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>DEPARTMENTS</td>
<td>DEPARTMENT_NAME</td>
<td>VARCHAR2</td>
<td>30</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>DEPARTMENTS</td>
<td>MANAGER_ID</td>
<td>NUMBER</td>
<td>22</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>DEPARTMENTS</td>
<td>LOCATION_ID</td>
<td>NUMBER</td>
<td>22</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>EMPLOYEES</td>
<td>EMPLOYEE_ID</td>
<td>NUMBER</td>
<td>22</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>EMPLOYEES</td>
<td>FIRST_NAME</td>
<td>VARCHAR2</td>
<td>20</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>EMPLOYEES</td>
<td>LAST_NAME</td>
<td>VARCHAR2</td>
<td>25</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>EMPLOYEES</td>
<td>EMAIL</td>
<td>VARCHAR2</td>
<td>25</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>LOCATIONS</td>
<td>COUNTRY_ID</td>
<td>CHAR</td>
<td>2</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>REGIONS</td>
<td>REGION_ID</td>
<td>NUMBER</td>
<td>22</td>
<td>05-FEB-03</td>
</tr>
<tr>
<td>REGIONS</td>
<td>REGION_NAME</td>
<td>VARCHAR2</td>
<td>25</td>
<td>05-FEB-03</td>
</tr>
</tbody>
</table>

51 rows selected.
13. Indexes Management

13.1. About Indexes
Indexes are optional structures associated with tables and clusters that allow SQL statements to execute more quickly against a table. Just as the index in this manual helps you locate information faster than if there were no index, an Oracle Database index provides a faster access path to table data. You can use indexes without rewriting any queries. Your results are the same, but you see them more quickly. Oracle Database provides several indexing schemes that provide complementary performance functionality. These are:

- B-tree indexes: the default and the most common
- B-tree cluster indexes: defined specifically for cluster
- Hash cluster indexes: defined specifically for a hash cluster
- Global and local indexes: relate to partitioned tables and indexes
- Reverse key indexes: most useful for Oracle Real Application Clusters applications
- Bitmap indexes: compact; work best for columns with a small set of values
- Function-based indexes: contain the precomputed value of a function/expression
- Domain indexes: specific to an application or cartridge.

Indexes are logically and physically independent of the data in the associated table. Being independent structures, they require storage space. You can create or drop an index without affecting the base tables, database applications, or other indexes. The database automatically maintains indexes when you insert, update, and delete rows of the associated table. If you drop an index, all applications continue to work. However, access to previously indexed data might be slower.

13.2. Guidelines for Managing Indexes

13.2.1. Create Indexes after Inserting Table Data
Data is often inserted or loaded into a table using either the SQL*Loader or an import utility. It is more efficient to create an index for a table after inserting or loading the data. If you create one or more indexes before loading data, the database then must update every index as each row is inserted. Creating an index on a table that already has data requires sort space. Some sort space comes from memory allocated for the index creator. The amount for each user is determined by the initialization parameter SORT_AREA_SIZE. The database also swaps sort information to and from temporary segments that are only allocated during the index creation in the users’ temporary tablespace. Under certain conditions, data can be loaded into a table with SQL*Loader direct-path load and an index can be created as data is loaded.

13.2.2. Index the Correct Tables and Columns
Use the following guidelines for determining when to create an index:

- Create an index if you frequently want to retrieve less than 15% of the rows in a large table. The percentage varies greatly according to the relative speed of a table scan and how the distribution of the row data in relation to the index key. The faster the table scans, the lower the percentage; the more clustered the row data, the higher the percentage.
- To improve performance on joins of multiple tables, index columns used for joins.

Note: Primary and unique keys automatically have indexes, but you might want to create an index on a foreign key. Small tables do not require indexes. If a query is taking too long, then the table might have grown from small to large.
13.2.3. Columns That Are Suitable for Indexing

Some columns are strong candidates for indexing. Columns with one or more of the following characteristics are candidates for indexing:

- Values are relatively unique in the column.
- There is a wide range of values (good for regular indexes).
- There is a small range of values (good for bitmap indexes).
- The column contains many nulls, but queries often select all rows having a value.

In this case, use the following phrase:

\[
\text{WHERE COL}_X > -9.99 \times \text{power}(10,125)
\]

Using the preceding phrase is preferable to:

\[
\text{WHERE COL}_X \text{ IS NOT NULL}
\]

This is because the first uses an index on COL\_X (assuming that COL\_X is a numeric column).

13.2.4. Order Index Columns for Performance

The order of columns in the CREATE INDEX statement can affect query performance. In general, specify the most frequently used columns first. If you create a single index across columns to speed up queries that access, for example, col1, col2, and col3; then queries that access just col1, or that access just col1 and col2, are also speeded up. But a query that accessed just col2, just col3, or just col2 and col3 does not use the index.

13.2.5. Limit the Number of Indexes for Each Table

A table can have any number of indexes. However, the more indexes there are, the more overhead is incurred as the table is modified. Specifically, when rows are inserted or deleted, all indexes on the table must be updated as well. Also, when a column is updated, all indexes that contain the column must be updated. Thus, there is a trade-off between the speed of retrieving data from a table and the speed of updating the table. For example, if a table is primarily read-only, having more indexes can be useful; but if a table is heavily updated, having fewer indexes could be preferable.

13.2.6. Drop Indexes That Are No Longer Required

Consider dropping an index if:

- It does not speed up queries. The table could be very small, or there could be many rows in the table but very few index entries.
- The queries in your applications do not use the index.
- The index must be dropped before being rebuilt.

13.2.7. Estimate Index Size and Set Storage Parameters

Estimating the size of an index before creating one can facilitate better disk space planning and management. You can use the combined estimated size of indexes, along with estimates for tables, the undo tablespace, and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases and other decisions. Use the estimated size of an individual index to
better manage the disk space that the index uses. When an index is created, you can set appropriate storage parameters and improve I/O performance of applications that use the index. For example, assume that you estimate the maximum size of an index before creating it. If you then set the storage parameters when you create the index, fewer extents are allocated for the table data segment, and all of the index data is stored in a relatively contiguous section of disk space. This decreases the time necessary for disk I/O operations involving this index. The maximum size of a single index entry is approximately one-half the data block size.

13.2.8. Specify the Tablespace for Each Index

Indexes can be created in any tablespace. An index can be created in the same or different tablespace as the table it indexes. If you use the same tablespace for a table and its index, it can be more convenient to perform database maintenance (such as tablespace or file backup) or to ensure application availability. All the related data is always online together. Using different tablespaces (on different disks) for a table and its index produces better performance than storing the table and index in the same tablespace. Disk contention is reduced. But, if you use different tablespaces for a table and its index and one tablespace is offline (containing either data or index), then the statements referencing that table are not guaranteed to work.

13.2.9. Consider Parallelizing Index Creation

You can parallelize index creation, much the same as you can parallelize table creation. Because multiple processes work together to create the index, the database can create the index more quickly than if a single server process created the index sequentially. When creating an index in parallel, storage parameters are used separately by each query server process. Therefore, an index created with an INITIAL value of 5M and a parallel degree of 12 consumes at least 60M of storage during index creation.

13.2.10. Consider Creating Indexes with NOLOGGING

You can create an index and generate minimal redo log records by specifying NOLOGGING in the CREATE INDEX statement.

Note: Because indexes created using NOLOGGING are not archived, perform a backup after you create the index.

Creating an index with NOLOGGING has the following benefits:

- Space is saved in the redo log files.
- The time it takes to create the index is decreased.
- Performance improves for parallel creation of large indexes.

In general, the relative performance improvement is greater for larger indexes created without LOGGING than for smaller ones. Creating small indexes without LOGGING has little effect on the time it takes to create an index. However, for larger indexes the performance improvement can be significant, especially when you are also parallelizing the index creation.

13.2.11. Consider Costs and Benefits of Coalescing or Rebuilding Indexes

Improper sizing or increased growth can produce index fragmentation. To eliminate or reduce fragmentation, you can rebuild or coalesce the index. But before you perform either task weigh the costs and benefits of each option and choose the one that works best for your situation. Table is a comparison of the costs and benefits associated with rebuilding and coalescing indexes.

<table>
<thead>
<tr>
<th>Rebuild Index</th>
<th>Coalesce Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quickly moves index to another tablespace</td>
<td>Cannot move index to another tablespace</td>
</tr>
<tr>
<td>Higher costs: requires more disk space</td>
<td>Lower costs: does not require more disk space</td>
</tr>
<tr>
<td>Creates new tree, shrinks height if applicable</td>
<td>Coalesces leaf blocks within same branch of tree</td>
</tr>
<tr>
<td>Enables you to quickly change storage and tablespace parameters without having to drop the original index.</td>
<td>Quickly frees up index leaf blocks for use.</td>
</tr>
</tbody>
</table>
In situations where you have B-tree index leaf blocks that can be freed up for reuse, you can merge those leaf blocks using the following statement:

```
ALTER INDEX vmoo re COALESCE;
```

Figure illustrates the effect of an ALTER INDEX COALESCE on the index vmoore. Before performing the operation, the first two leaf blocks are 50% full. This means you have an opportunity to reduce fragmentation and completely fill the first block, while freeing up the second.

**Figure: Coalescing Indexes**

13.2.12. Consider Cost before Disabling or Dropping Constraints

Because unique and primary keys have associated indexes, you should factor in the cost of dropping and creating indexes when considering whether to disable or drop a UNIQUE or PRIMARY KEY constraint. If the associated index for a UNIQUE key or PRIMARY KEY constraint is extremely large, you can save time by leaving the constraint enabled rather than dropping and re-creating the large index. You also have the option of explicitly specifying that you want to keep or drop the index when dropping or disabling a UNIQUE or PRIMARY KEY constraint.

13.3. Creating Indexes

This section describes how to create indexes. To create an index in your own schema, at least one of the following conditions must be true:

- The table or cluster to be indexed is in your own schema.
- You have INDEX privilege on the table to be indexed.
- You have CREATE ANY INDEX system privilege.

To create an index in another schema, all of the following conditions must be true:

- You have CREATE ANY INDEX system privilege.
- The owner of the other schema has a quota for the tablespaces to contain the index or index partitions, or UNLIMITED TABLESPACE system privilege.

13.3.1. Creating an Index Explicitly

You can create indexes explicitly (outside of integrity constraints) using the SQL statement CREATE INDEX. The following statement creates an index named emp_ename for the ename column of the emp table:

```
CREATE INDEX emp_ename ON emp (ename)
  TABLESPACE users
  STORAGE (INITIAL 20K
           NEXT 20K
           PCTINCREASE 75);
```
Notice that several storage settings and a tablespace are explicitly specified for the index. If you do not specify storage options (such as INITIAL and NEXT) for an index, the default storage options of the default or specified tablespace are automatically used.

### 13.3.2. Creating a Unique Index Explicitly

Indexes can be unique or non-unique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns). Non-unique indexes do not impose this restriction on the column values. Use the `CREATE UNIQUE INDEX` statement to create a unique index. The following example creates a unique index:

```sql
CREATE UNIQUE INDEX dept_unique_index ON dept (dname) TABLESPACE indx;
```

Alternatively, you can define UNIQUE integrity constraints on the desired columns. The database enforces UNIQUE integrity constraints by automatically defining a unique index on the unique key. This is discussed in the following section. However, it is advisable that any index that exists for query performance, including unique indexes, be created explicitly.

### 13.3.3. Creating an Index Associated with a Constraint

Oracle Database enforces a UNIQUE key or PRIMARY KEY integrity constraint on a table by creating a unique index on the unique key or primary key. This index is automatically created by the database when the constraint is enabled. No action is required by you when you issue the `CREATE TABLE` or `ALTER TABLE` statement to create the index, but you can optionally specify a USING INDEX clause to exercise control over its creation. This includes both when a constraint is defined and enabled, and when a defined but disabled constraint is enabled. To enable a UNIQUE or PRIMARY KEY constraint, thus creating an associated index, the owner of the table must have a quota for the tablespace intended to contain the index, or the UNLIMITED Tablespace system privilege. The index associated with a constraint always takes the name of the constraint, unless you optionally specify otherwise.

**Note:** An efficient procedure for enabling a constraint that can make use of parallelism.

### 13.3.4. Specifying Storage Options for an Index Associated with a Constraint

You can set the storage options for the indexes associated with UNIQUE and PRIMARY KEY constraints using the USING INDEX clause. The following CREATE TABLE statement enables a PRIMARY KEY constraint and specifies the storage options of the associated index:

```sql
CREATE TABLE emp (empno NUMBER(5) PRIMARY KEY, age INTEGER) ENABLE PRIMARY KEY USING INDEX TABLESPACE users;
```

### 13.3.5. Specifying the Index Associated with a Constraint

If you require more explicit control over the indexes associated with UNIQUE and PRIMARY KEY constraints, the database lets you:

- Specify an existing index that the database is to use to enforce the constraint
- Specify a CREATE INDEX statement that the database is to use to create the index and enforce the constraint
- These options are specified using the USING INDEX clause. The following statements present some examples.

**Example 1:**

```sql
CREATE TABLE a (  
a1 INT PRIMARY KEY USING INDEX (create index ai on a (a1)));
```
Example 2:
```
CREATE TABLE b(
  b1 INT,
  b2 INT,
  CONSTRAINT bu1 UNIQUE (b1, b2)
    USING INDEX (create unique index bi on b(b1, b2)),
  CONSTRAINT bu2 UNIQUE (b2, b1) USING INDEX bi);
```

Example 3:
```
CREATE TABLE c(c1 INT, c2 INT);
CREATE INDEX ci ON c (c1, c2);
ALTER TABLE c ADD CONSTRAINT cpk PRIMARY KEY (c1) USING INDEX ci;
```

If a single statement creates an index with one constraint and also uses that index for another constraint, the system will attempt to rearrange the clauses to create the index before reusing it. Oracle Database provides you with the opportunity to collect statistics at very little resource cost during the creation or rebuilding of an index. These statistics are stored in the data dictionary for ongoing use by the optimizer in choosing a plan for the execution of SQL statements. The following statement computes index, table, and column statistics while building index emp_ename on column ename of table emp:

```
CREATE INDEX emp_ename ON emp(ename) COMPUTE STATISTICS;
```

### 13.3.6. Creating an Index Online

You can create and rebuild indexes online. This enables you to update base tables at the same time you are building or rebuilding indexes on that table. You can perform DML operations while the index build is taking place, but DDL operations are not allowed. Parallel execution is not supported when creating or rebuilding an index online. The following statements illustrate online index build operations:

```
CREATE INDEX emp_name ON emp (mgr, emp1, emp2, emp3) ONLINE;
```

Note: Keep in mind that the time that it takes on online index build to complete is proportional to the size of the table and the number of concurrently executing DML statements. Therefore, it is best to start online index builds when DML activity is low.

### 13.3.7. Creating a Function-Based Index

Function-based indexes facilitate queries that qualify a value returned by a function or expression. The value of the function or expression is pre-computed and stored in the index. In addition to the prerequisites for creating a conventional index, if the index is based on user-defined functions, then those functions must be marked DETERMINISTIC. Also, you just have the EXECUTE object privilege on any user-defined function(s) used in the function-based index if those functions are owned by another user. Additionally, to use a function-based index:

- The table must be analyzed after the index is created.
- The query must be guaranteed not to need any NULL values from the indexed expression, since NULL values are not stored in indexes.

**Note:** CREATE INDEX stores the timestamp of the most recent function used in the function-based index. This timestamp is updated when the index is validated. When performing tablespace point-in-time recovery of a function-based index, if the timestamp on the most recent function used in the index is newer than the timestamp stored in the index, then the index is marked invalid. You must use the ANALYZE INDEX...VALIDATE STRUCTURE statement to validate this index.

To illustrate a function-based index, consider the following statement that defines a function-based index (area_index) defined on the function area(geo):

```
CREATE INDEX area_index ON rivers (area(geo));
```
In the following SQL statement, when area(geo) is referenced in the WHERE clause, the optimizer considers using the index area_index.

```
SELECT id, geo, area(geo), desc FROM rivers WHERE Area(geo) >5000;
```

Table owners should have EXECUTE privileges on the functions used in function-based indexes. Because a function-based index depends upon any function it is using, it can be invalidated when a function changes. If the function is valid, you can use an ALTER INDEX...ENABLE statement to enable a function-based index that has been disabled. The ALTER INDEX...DISABLE statement lets you disable the use of a function-based index. Consider doing this if you are working on the body of the function.

**Note:** An alternative to creating a function-based index is to add a virtual column to the target table and index the virtual column.

### 13.3.8. Creating a Key-Compressed Index

Creating an index using key compression enables you to eliminate repeated occurrences of key column prefix values. Key compression breaks an index key into a prefix and a suffix entry. Compression is achieved by sharing the prefix entries among all the suffix entries in an index block. This sharing can lead to huge savings in space, allowing you to store more keys for each index block while improving performance. Key compression can be useful in the following situations:

- You have a non-unique index where ROWID is appended to make the key unique. If you use key compression here, the duplicate key is stored as a prefix entry on the index block without the ROWID. The remaining rows become suffix entries consisting of only the ROWID.
- You have a unique multicolumn index.

You enable key compression using the COMPRESS clause. The prefix length (as the number of key columns) can also be specified to identify how the key columns are broken into a prefix and suffix entry. For example, the following statement compresses duplicate occurrences of a key in the index leaf block:

```
CREATE INDEX emp_ename ON emp(ename) TABLESPACE users COMPRESS 1;
```

You can also specify the COMPRESS clause during rebuild. For example, during rebuild you can disable compression as follows:

```
ALTER INDEX emp_ename REBUILD NOCOMPRESS;
```

### 13.3.9. Creating an Invisible Index

Beginning with Release 11g, you can create invisible indexes. An invisible index is an index that is ignored by the optimizer unless you explicitly set the OPTIMIZER_USE_INVISIBLE_INDEXES initialization parameter to TRUE at the session or system level. Making an index invisible is an alternative to making it unusable or dropping it. Using invisible indexes, you can do the following:

- Test the removal of an index before dropping it.
- Use temporary index structures for certain operations or modules of an application without affecting the overall application.

Unlike unusable indexes, an invisible index is maintained during DML statements. To create an invisible index, use the SQL statement CREATE INDEX with the INVISIBLE clause. The following statement creates an invisible index named emp_ename for the ename column of the emp table:

```
CREATE INDEX emp_ename ON emp(ename) TABLESPACE users STORAGE (INITIAL 20K NEXT 20k PCTINCREASE 75) INVISIBLE;
```
13.4. Altering Indexes

To alter an index, your schema must contain the index or you must have the ALTER ANY INDEX system privilege. With the ALTER INDEX statement, you can:

- Rebuild or coalesce an existing index
- Deallocate unused space or allocate a new extent
- Specify parallel execution (or not) and alter the degree of parallelism
- Alter storage parameters or physical attributes
- Specify LOGGING or NOLOGGING
- Enable or disable key compression
- Mark the index unusable
- Make the index invisible
- Start or stop the monitoring of index usage

13.4.1. Altering Storage Characteristics of an Index

Alter the storage parameters of any index, including those created by the database to enforce primary and unique key integrity constraints, using the ALTER INDEX statement. For example, the following statement alters the emp_ename index:

```
ALTER INDEX emp_ename STORAGE (PCTINCREASE 50);
```

The storage parameters INITIAL and MINEXTENTS cannot be altered. All new settings for the other storage parameters affect only extents subsequently allocated for the index. For indexes that implement integrity constraints, you can adjust storage parameters by issuing an ALTER TABLE statement that includes the USING INDEX subclause of the ENABLE clause. For example, the following statement changes the storage options of the index created on table emp to enforce the primary key constraint:

```
ALTER TABLE emp ENABLE PRIMARY KEY USING INDEX;
```

13.4.2. Rebuilding an Existing Index

Before rebuilding an existing index, compare the costs and benefits associated with rebuilding to those associated with coalescing indexes. When you rebuild an index, you use an existing index as the data source. Creating an index in this manner enables you to change storage characteristics or move to a new tablespace. Rebuilding an index based on an existing data source removes intra-block fragmentation. Compared to dropping the index and using the CREATE INDEX statement, re-creating an existing index offers better performance. The following statement rebuilds the existing index emp_name:

```
ALTER INDEX emp_name REBUILD;
```

The REBUILD clause must immediately follow the index name, and precede any other options. It cannot be used in conjunction with the DEALLOCATE UNUSED clause. You have the option of rebuilding the index online. Rebuilding online enables you to update base tables at the same time that you are rebuilding. The following statement rebuilds the emp_name index online:

```
ALTER INDEX emp_name REBUILD ONLINE;
```

Note: Online index rebuilding has stricter limitations on the maximum key length that can be handled, compared to other methods of rebuilding an index. If an ORA-1450 (maximum key length exceeded) error occurs when rebuilding online, try rebuilding offline, coalescing, or dropping and recreating the index.

If you do not have the space required to rebuild an index, you can choose instead to coalesce the index. Coalescing an index is an online operation.
13.4.3. Making an Index Invisible
To make a visible index invisible, issue this statement:

```
ALTER INDEX index INVISIBLE;
```

To make an invisible index visible, issue this statement:

```
ALTER INDEX index VISIBLE;
```

To find out whether an index is visible or invisible, query the dictionary views USER_INDEXES, ALL_INDEXES, or DBA_INDEXES. For example, to determine if the index IND1 is invisible, issue the following query:

```
SELECT INDEX_NAME, VISIBILITY FROM USER_INDEXES WHERE INDEX_NAME = 'IND1';
```

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>VISIBILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>IND1</td>
<td>VISIBLE</td>
</tr>
</tbody>
</table>

13.4.4. Monitoring Index Usage
Oracle Database provides a means of monitoring indexes to determine whether they are being used. If an index is not being used, then it can be dropped, eliminating unnecessary statement overhead.

To start monitoring the usage of an index, issue this statement:

```
ALTER INDEX index MONITORING USAGE;
```

Later, issue the following statement to stop the monitoring:

```
ALTER INDEX index NOMONITORING USAGE;
```

The view V$OBJECT_USAGE can be queried for the index being monitored to see if the index has been used. The view contains a USED column whose value is YES or NO, depending upon if the index has been used within the time period being monitored. The view also contains the start and stop times of the monitoring period, and a MONITORING column (YES/NO) to indicate if usage monitoring is currently active.

Each time that you specify MONITORING USAGE, the V$OBJECT_USAGE view is reset for the specified index. The previous usage information is cleared or reset, and a new start time is recorded. When you specify NOMONITORING USAGE, no further monitoring is performed, and the end time is recorded for the monitoring period. Until the next ALTER INDEX...MONITORING USAGE statement is issued, the view information is left unchanged.

13.4.5. Monitoring Space Use of Indexes
If key values in an index are inserted, updated, and deleted frequently, the index can lose its acquired space efficiently over time. Monitor index efficiency of space usage at regular intervals by first analyzing the index structure, using the ANALYZE INDEX...VALIDATE STRUCTURE statement, and then querying the INDEX_STATS view:

```
SELECT PCT_USED FROM INDEX_STATS WHERE NAME = 'index';
```

The percentage of index space usage varies according to how often index keys are inserted, updated, or deleted. Develop a history of average efficiency of space usage for an index by performing the following sequence of operations several times:

- Analyzing statistics
- Validating the index
- Checking PCT_USED
- Dropping and rebuilding (or coalescing) the index
When you find that index space usage drops below its average, you can condense the index space by dropping the index and rebuilding it, or coalescing it. To drop an index, the index must be contained in your schema, or you must have the DROP ANY INDEX system privilege. Some reasons for dropping an index include:

- The index is no longer required.
- The index is not providing anticipated performance improvements for queries issued against the associated table. For example, the table might be very small, or there might be many rows in the table but very few index entries.
- Applications do not use the index to query the data.
- The index has become invalid and must be dropped before being rebuilt.
- The index has become too fragmented and must be dropped before being rebuilt.

When you drop an index, all extents of the index segment are returned to the containing tablespace and become available for other objects in the tablespace. How you drop an index depends on whether you created the index explicitly with a CREATE INDEX statement, or implicitly by defining a key constraint on a table. If you created the index explicitly with the CREATE INDEX statement, then you can drop the index with the DROP INDEX statement. The following statement drops the emp_ename index:

```
DROP INDEX emp_ename;
```

You cannot drop only the index associated with an enabled UNIQUE key or PRIMARY KEY constraint. To drop a constraints associated index, you must disable or drop the constraint itself.

**Note:** If a table is dropped, all associated indexes are dropped automatically.

### 13.4.6 Indexes Data Dictionary Views

The following views display information about indexes:

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA_INDEXES</td>
<td>DBA view describes indexes on all tables in the database. All view describes indexes on all tables accessible to the user. User view is restricted to indexes owned by the user. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.</td>
</tr>
<tr>
<td>ALL_INDEXES</td>
<td>These views describe the columns of indexes on tables. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.</td>
</tr>
<tr>
<td>USER_INDEXES</td>
<td>These views describe the expressions of function-based indexes on tables.</td>
</tr>
<tr>
<td>DBA_IND_COLUMNS</td>
<td>These views contain optimizer statistics for indexes.</td>
</tr>
<tr>
<td>ALL_IND_COLUMNS</td>
<td>Stores information from the last ANALYZE INDEX...VALIDATE STRUCTURE statement.</td>
</tr>
<tr>
<td>USER_IND_COLUMNS</td>
<td>Stores information from the last ANALYZE INDEX...VALIDATE STRUCTURE statement.</td>
</tr>
<tr>
<td>DBA_IND_EXPRESSIONS</td>
<td>Contains index usage information produced by the ALTER INDEX...MONITORING USAGE functionality.</td>
</tr>
<tr>
<td>ALL_IND_EXPRESSIONS</td>
<td></td>
</tr>
<tr>
<td>USER_IND_EXPRESSIONS</td>
<td></td>
</tr>
</tbody>
</table>
14. SQL*Loader

SQL*Loader loads data from external files into tables of an Oracle database. It has a powerful data parsing engine that puts little limitation on the format of the data in the datafile. You can use SQL*Loader to do the following:

- Load data across a network if your data files are on a different system than the database.
- Load data from multiple datafiles during the same load session.
- Load data into multiple tables during the same load session.
- Specify the character set of the data.
- Selectively load data (you can load records based on the records' values).
- Manipulate the data before loading it, using SQL functions.
- Generate unique sequential key values in specified columns.
- Use the operating system's file system to access the datafiles.
- Load data from disk, tape, or named pipe.
- Generate sophisticated error reports, which greatly aid troubleshooting.
- Load arbitrarily complex object-relational data.
- Use secondary datafiles for loading LOBs and collections.
- Use either conventional or direct path loading. While conventional path loading is very flexible, direct path loading provides superior loading performance.

A typical SQL*Loader session takes as input a control file, which controls the behavior of SQL*Loader, and one or more datafiles. The output of SQL*Loader is an Oracle database (where the data is loaded), a log file, a bad file, and potentially, a discard file. An example of the flow of a SQL*Loader session is shown in Figure.

14.1. SQL*Loader Parameters

SQL*Loader is invoked when you specify the sqlldr command and, optionally, parameters that establish session characteristics. In situations where you always use the same parameters for which the values seldom change, it can be more efficient to specify parameters using the following methods, rather than on the command line:

- Parameters can be grouped together in a parameter file. You could then specify the name of the parameter file on the command line using the PARFILE parameter.
• Certain parameters can also be specified within the SQL*Loader control file by using the OPTIONS clause.

Parameters specified on the command line override any parameter values specified in a parameter file or OPTIONS clause.

14.2. SQL*Loader Control File

The control file is a text file written in a language that SQL*Loader understands. The control file tells SQL*Loader where to find the data, how to parse and interpret the data, where to insert the data, and more. Although not precisely defined, a control file can be said to have three sections. The first section contains session-wide information, for example:

• Global options such as bindsize, rows, records to skip, and so on
• INFILE clauses to specify where the input data is located
• Data to be loaded

The second section consists of one or more INTO TABLE blocks. Each of these blocks contains information about the table into which the data is to be loaded, such as the table name and the columns of the table. The third section is optional and, if present, contains input data. Some control file syntax considerations to keep in mind are:

• The syntax is free-format (statements can extend over multiple lines).
• It is case insensitive; however, strings enclosed in single or double quotation marks are taken literally, including case.
• In control file syntax, comments extend from the two hyphens (--) that mark the beginning of the comment to the end of the line. The optional third section of the control file is interpreted as data rather than as control file syntax; consequently, comments in this section are not supported.
• The keywords CONSTANT and ZONE have special meaning to SQL*Loader and are therefore reserved. To avoid potential conflicts, Oracle recommends that you do not use either CONSTANT or ZONE as a name for any tables or columns.

14.3. Input Data and Datafiles

SQL*Loader reads data from one or more files (or operating system equivalents of files) specified in the control file. From SQL*Loader's perspective, the data in the datafile is organized as records. A particular datafile can be in fixed record format, variable record format, or stream record format. The record format can be specified in the control file with the INFILE parameter. If no record format is specified, the default is stream record format.

Note: If data is specified inside the control file (that is, INFILE * was specified in the control file), then the data is interpreted in the stream record format with the default record terminator.

14.3.1. Fixed Record Format

A file is in fixed record format when all records in a datafile are the same byte length. Although this format is the least flexible, it results in better performance than variable or stream format. Fixed format is also simple to specify. For example:

```
INFILE datafile_name "fix n"
```

This example specifies that SQL*Loader should interpret the particular datafile as being in fixed record format where every record is n bytes long. Example shows a control file that specifies a datafile that should be interpreted in the fixed record format. The datafile in the example contains five physical records. Assuming that a period (.) indicates a space, the first physical record is [001,...cd,...] which is exactly eleven bytes (assuming a single-byte character set). The second record is [0002,fghi,\n] followed by the newline character (which is the eleventh byte),
and so on. Note that newline characters are not required with the fixed record format. Note that the length is always interpreted in bytes, even if character-length semantics are in effect for the file. This is necessary because the file could contain a mix of fields, some of which are processed with character-length semantics and others which are processed with byte-length semantics.

**Example Loading Data in Fixed Record Format**

```sql
load data
infile 'example.dat' "fix 11"
into table example
fields terminated by ',' optionally enclosed by ""
(coll, col2)
example.dat:
001, cd, 0002, fghi,
00003, lmn,
1, "pqrs",
0005, uvwx,
```

14.3.2. Variable Record Format

A file is in variable record format when the length of each record in a character field is included at the beginning of each record in the datafile. This format provides some added flexibility over the fixed record format and a performance advantage over the stream record format. For example, you can specify a datafile that is to be interpreted as being in variable record format as follows:

```sql
INFILE "datafile_name" "var n"
```

In this example, n specifies the number of bytes in the record length field. If n is not specified, SQL*Loader assumes a length of 5 bytes. Specifying n larger than 40 will result in an error. Example shows a control file specification that tells SQL*Loader to look for data in the datafile example.dat and to expect variable record format where the record length fields are 3 bytes long. The example.dat datafile consists of three physical records. The first is specified to be 009 (that is, 9) bytes long, the second is 010 bytes long (that is, 10, including a 1-byte newline), and the third is 012 bytes long (also including a 1-byte newline). Note that newline characters are not required with the variable record format. This example also assumes a single-byte character set for the datafile.

The lengths are always interpreted in bytes, even if character-length semantics are in effect for the file. This is necessary because the file could contain a mix of fields, some processed with character-length semantics and others processed with byte-length semantics.

**Loading Data in Variable Record Format**

```sql
load data
infile 'example.dat' "var 3"
into table example
fields terminated by ',' optionally enclosed by ""
(coll char(5),
col2 char(7))
example.dat:
009hello, cd, 010world, im,
012my, name is,
```

14.4. LOBFILEs and Secondary Datafiles (SDFs)

LOB data can be lengthy enough that it makes sense to load it from a LOBFILE. In LOBFILEs, LOB data instances are still considered to be in fields (predetermined size, delimited, length-value), but these fields are not organized into records (the concept of a record does not exist within LOBFILEs). Therefore, the processing overhead of dealing with records is avoided. This type of organization of data is ideal for LOB loading. For example, you might
use LOBFILEs to load employee names, employee IDs, and employee resumes. You could read the employee names and IDs from the main datafiles and you could read the resumes, which can be quite lengthy, from LOBFILEs. You might also use LOBFILEs to facilitate the loading of XML data. You can use XML columns to hold data that models structured and semistructured data. Such data can be quite lengthy.

Secondary datafiles (SDFs) are similar in concept to primary datafiles. Like primary datafiles, SDFs are a collection of records, and each record is made up of fields. The SDFs are specified on a per control-file-field basis. Only a collection_fld_spec can name an SDF as its data source. SDFs are specified using the SDF parameter. The SDF parameter can be followed by either the file specification string, or a FILLER field that is mapped to a data field containing one or more file specification strings.

14.5. Data Conversion and Datatype Specification

During a conventional path load, data fields in the datafile are converted into columns in the database (direct path loads are conceptually similar, but the implementation is different). There are two conversion steps:

1. SQL*Loader uses the field specifications in the control file to interpret the format of the datafile, parse the input data, and populate the bind arrays that correspond to a SQL INSERT statement using that data.
2. The Oracle database accepts the data and executes the INSERT statement to store the data in the database.

The Oracle database uses the datatype of the column to convert the data into its final, stored form. Keep in mind the distinction between a field in a datafile and a column in the database. Remember also that the field datatypes defined in a SQL*Loader control file are not the same as the column datatypes.

14.6. Discarded and Rejected Records

Records read from the input file might not be inserted into the database. Such records are placed in either a bad file or a discard file.

14.6.1. The Bad File

The bad file contains records that were rejected, either by SQL*Loader or by the Oracle database. If you do not specify a bad file and there are rejected records, then SQL*Loader automatically creates one. It will have the same name as the data file, with a .bad extension. Some of the possible reasons for rejection are discussed in the next sections.

14.6.2. SQL*Loader Rejects

Datafile records are rejected by SQL*Loader when the input format is invalid. For example, if the second enclosure delimiter is missing, or if a delimited field exceeds its maximum length, SQL*Loader rejects the record. Rejected records are placed in the bad file.

14.6.3. Oracle Database Rejects

After a datafile record is accepted for processing by SQL*Loader, it is sent to the Oracle database for insertion into a table as a row. If the Oracle database determines that the row is valid, then the row is inserted into the table. If the row is determined to be invalid, then the record is rejected and SQL*Loader puts it in the bad file. The row may be invalid, for example, because a key is not unique, because a required field is null, or because the field contains invalid data for the Oracle datatype.

14.6.4. The Discard File

As SQL*Loader executes, it may create a file called the discard file. This file is created only when it is needed, and only if you have specified that a discard file should be enabled. The discard file contains records that were filtered out of the load because they did not match any record-selection criteria specified in the control file. The discard file therefore contains records that were not inserted into any table in the database. You can specify the maximum number of such records that the discard file can accept. Data written to any database table is not written to the discard file.
14.7. Log File and Logging Information

When SQL*Loader begins execution, it creates a log file. If it cannot create a log file, execution terminates. The log file contains a detailed summary of the load, including a description of any errors that occurred during the load.


SQL*Loader provides the following methods to load data:

- Conventional Path Loads
- Direct Path Loads


During conventional path loads, the input records are parsed according to the field specifications, and each data field is copied to its corresponding bind array. When the bind array is full (or no more data is left to read), an array insert is executed. SQL*Loader stores LOB fields after a bind array insert is done. Thus, if there are any errors in processing the LOB field (for example, the LOBFILE could not be found), the LOB field is left empty. Note also that because LOB data is loaded after the array insert has been performed, BEFORE and AFTER row triggers may not work as expected for LOB columns. This is because the triggers fire before SQL*Loader has a chance to load the LOB contents into the column. For instance, suppose you are loading a LOB column, C1, with data and that you want a BEFORE row trigger to examine the contents of this LOB column and derive a value to be loaded for some other column, C2, based on its examination. This is not possible because the LOB contents will not have been loaded at the time the trigger fires.

14.8.2. Direct Path Loads

A direct path load parses the input records according to the field specifications, converts the input field data to the column datatype, and builds a column array. The column array is passed to a block formatter, which creates data blocks in Oracle database block format. The newly formatted database blocks are written directly to the database, bypassing much of the data processing that normally takes place. Direct path load is much faster than conventional path load, but entails several restrictions.

14.8.3. Parallel Direct Path

A parallel direct path load allows multiple direct path load sessions to concurrently load the same data segments (allows intrasegment parallelism). Parallel direct path is more restrictive than direct path.
INDEX

1. Introduction to Database Administration ................................................................. 1
   1.1. Types of Oracle Database Users ........................................................................ 1
   1.2. Tasks of a Database Administrator .................................................................. 2
   1.3. About SQL*Plus ............................................................................................... 4
      1.3.1. Connecting to the Database with SQL*Plus ................................................ 4
   1.4. Release Number Format ................................................................................... 7
      1.4.1. Checking Your Current Release Number ...................................................... 8
   1.5. About Database Administrator Security and Privileges ................................... 8
      1.5.1. The Database Administrator's Operating System Account ......................... 8
      1.5.2. Administrative User Accounts ................................................................... 8
      1.5.3. The DBA Role ............................................................................................ 9
      1.5.4. Database Administrator Authentication ...................................................... 9
      1.5.5. Authentication Method for Database Administrators .................................. 10
      1.5.6. Using Operating System Authentication ..................................................... 12
      1.5.7. Preparing to Use Operating System Authentication ..................................... 12
      1.5.8. Connecting Using Password File Authentication ........................................ 13
      1.5.9. Creating and Maintaining a Password File ................................................ 13
      1.5.10. Adding Users to a Password File ............................................................. 15
      1.5.11. Maintaining a Password File .................................................................... 16
   1.6. Data Utilities ................................................................................................... 16

2. Oracle Database Creation and Configuration ....................................................... 17
   2.1. About Creating an Oracle Database ................................................................. 17
   2.2. Considerations before Creating the Database .................................................. 17
      2.2.1. Meeting Creation Prerequisites .................................................................. 17
      2.2.2. Creating a Database with DBCA ................................................................ 17
   2.3. Understanding the CREATE DATABASE Statement ........................................... 23
      2.3.1. Creating a Locally Managed SYSTEM Tablespace ........................................ 24
      2.3.2. About the SYSAUX Tablespace .................................................................. 24
      2.3.3. Using Automatic Undo Management: Creating an Undo Tablespace ........... 24
      2.3.4. Creating a Default Permanent Tablespace .................................................. 25
      2.3.5. Creating a Default Temporary Tablespace ................................................... 25
   2.4. Specifying Oracle-Managed Files at Database Creation ................................... 25
      2.4.1. Supporting Bigfile Tablespaces during Database Creation ............................ 26
      2.4.2. Overriding the Default Tablespace Type ...................................................... 26
   2.5. Understanding Initialization Parameters .......................................................... 27
      2.5.1. Sample Initialization Parameter File ............................................................ 27
   2.6. Dropping a Database ....................................................................................... 31
   2.7. Managing Initialization Parameters Using a Server Parameter File ................. 31
      2.7.1. What Is a Server Parameter File? ................................................................. 32
      2.7.2. Migrating to a Server Parameter File .......................................................... 32
      2.7.3. Creating a Server Parameter File ................................................................. 32
      2.7.3. The SPFILE Initialization Parameter ........................................................... 34
3. Oracle Instance Memory Architecture............................................................................. ........................ 37

3.1. Basic Memory Structure......................................................................................... 37
3.1.1. System Global Area (SGA) .............................................................................. 37
3.1.2. Program Global Area (PGA)........................................................................... 37
3.1.3. Setting the Buffer Cache Initialization Parameters........................................... 37
3.1.4. Specifying the Shared Pool Size ..................................................................... 39
3.1.5. Specifying the Large Pool Size ...................................................................... 39
3.1.6. Specifying the Java Pool Size ......................................................................... 39
3.1.7. Specifying the Streams Pool Size ................................................................... 39
3.1.8. Specifying the Result Cache Maximum Size.................................................. 39

3.2. Using Automatic PGA Memory Management ......................................................... 40
3.2.1. Using Manual PGA Memory Management...................................................... 41

4. Users and Resource Management ............................................................................. 43
4.1. Managing User Privileges and Roles .................................................................. 43
4.2. Auditing Database Use ....................................................................................... 43
4.3. Creating a New User Account .......................................................................... 43
4.4. Assigning a Default Tablespace for the User ..................................................... 44
4.5. Assigning a Tablespace Quota for the User ....................................................... 45
4.6. Revoking the Ability for Users to Create Objects in a Tablespace....................... 46
4.6.1 Granting Users the UNLIMITED TABLESPACE System Privilege .................. 46
4.7. Assigning a Temporary Tablespace for the User ............................................... 46
4.8. Specifying a Profile for the User ....................................................................... 47
4.9. Setting a Default Role for the User ................................................................... 47
4.10. Altering User Accounts..................................................................................... 47
4.10.1. Changing the User Password .................................................................... 48
4.11. Types of System Resources and Limits ............................................................ 48
4.11.1. Limiting the User Session Level .................................................................. 48
4.11.2. Limiting Database Call Levels ................................................................... 48
4.11.3. Limiting CPU Time.................................................................................... 48
4.11.4. Limiting Logical Reads............................................................................. 49
4.11.5. Limiting Other Resources........................................................................... 49
4.11.6. Determining Values for Resource Limits of Profiles................................... 49
4.12. Managing Resources with Profiles ................................................................... 49
4.12.1. Creating Profiles....................................................................................... 50
4.12.2. Dropping Profiles...................................................................................... 50
4.12.3. Deleting User Accounts............................................................................. 50
4.12.4. Killing a User Session............................................................................... 51
4.12.5. Dropping a User Account........................................................................... 51
4.13. Listing All Users and Associated Information................................................... 52
4.13.1. Listing All Tablespace Quotas .................................................................... 52
4.13.2. Listing All Profiles and Assigned Limits.................................................... 52

5. Controlfile Management............................................................................................. 55
7. Archived Redo Logs Management ................................................................. 69

7.1. What is the Archived Redo Log? ................................................................. 69

7.2. Choosing Between NOARCHIVELOG and ARCHIVELOG Mode ................. 69

7.2.1. Running a Database in NOARCHIVELOG Mode ........................................ 69

7.2.2. Running a Database in ARCHIVELOG Mode ............................................. 70

7.3. Controlling Archiving .................................................................................. 70

7.3.1 Setting the Initial Database Archiving Mode .............................................. 70

5.4. Recovering a Control File Using a Current Copy ........................................... 58

5.4.1. Recovering from Control File Corruption Using a Control File Copy .......... 58

5.4.2. Recovering from Permanent Media Failure Using a Control File Copy ....... 59

5.5. Dropping Control Files ................................................................................ 59

5.3. Creating Control Files .................................................................................. 56

5.3.1. Creating Additional Copies, Renaming, and Relocating Control Files ....... 56

5.3.2. Creating New Control Files ..................................................................... 57

5.3.3. Steps for Creating New Control Files ..................................................... 58

5.4. Recovering a Control File Using a Current Copy ........................................... 58

5.4.1. Recovering from Control File Corruption Using a Control File Copy .......... 58

5.4.2. Recovering from Permanent Media Failure Using a Control File Copy ....... 59

5.5. Dropping Control Files ................................................................................ 59

5.2. Guidelines for Control Files ......................................................................... 55

5.2.1. Provide Filenames for the Control Files .................................................... 55

5.2.2. Multiplex Control Files on Different Disks .............................................. 55

5.2.3. Back Up Control Files ............................................................................ 56

5.2.4. Manage the Size of Control Files ............................................................ 56

5.3. Creating Control Files .................................................................................. 56

5.3.1. Creating Additional Copies, Renaming, and Relocating Control Files ....... 56

5.3.2. Creating New Control Files ..................................................................... 57

5.3.3. Steps for Creating New Control Files ..................................................... 58

5.4. Recovering a Control File Using a Current Copy ........................................... 58

5.4.1. Recovering from Control File Corruption Using a Control File Copy .......... 58

5.4.2. Recovering from Permanent Media Failure Using a Control File Copy ....... 59

5.5. Dropping Control Files ................................................................................ 59

5.1. What Is a Control File? ................................................................................ 55

5.2. Guidelines for Control Files ......................................................................... 55

5.2.1. Provide Filenames for the Control Files .................................................... 55

5.2.2. Multiplex Control Files on Different Disks .............................................. 55

5.2.3. Back Up Control Files ............................................................................ 56

5.2.4. Manage the Size of Control Files ............................................................ 56

5.3. Creating Control Files .................................................................................. 56

5.3.1. Creating Additional Copies, Renaming, and Relocating Control Files ....... 56

5.3.2. Creating New Control Files ..................................................................... 57

5.3.3. Steps for Creating New Control Files ..................................................... 58

5.4. Recovering a Control File Using a Current Copy ........................................... 58

5.4.1. Recovering from Control File Corruption Using a Control File Copy .......... 58

5.4.2. Recovering from Permanent Media Failure Using a Control File Copy ....... 59

5.5. Dropping Control Files ................................................................................ 59
9. Transportable Tablespace ........................................................................... 91
  9.1. Introduction to Transportable Tablespaces .............................................. 91
  9.2. Transporting Tablespaces between Databases .......................................... 91
  9.3. About Transporting Tablespaces across Platforms .................................. 91
  9.4. Limitations on Transportable Tablespace Use ......................................... 92
  9.5. Transport Scenario Source Database Target Database ................................ 93
  9.3. About Transporting Tablespaces across Platforms .................................. 91
  9.4. Limitations on Transportable Tablespace Use ......................................... 92
  9.5. Transport Scenario Source Database Target Database ................................ 93
    9.6.1. Type of File Number Description ....................................................... 98

8. Tablespaces Management ............................................................................. 76
  8.1. Using Multiple Tablespaces .................................................................... 76
  8.2. Creating Tablespaces ............................................................................. 76
    8.2.1. Creating a Locally Managed Tablespace ............................................. 77
    8.2.2. Specifying Segment Space Management in Locally Managed Tablespaces .. 78
  8.3. Bigfile Tablespaces ................................................................................. 78
    8.3.1. Creating a Bigfile Tablespace .............................................................. 79
  8.4. Encrypted Tablespaces ........................................................................... 80
    8.4.1. Restrictions ........................................................................................ 81
    8.4.2. Querying Tablespace Encryption Information ...................................... 81
  8.5. Temporary Tablespaces ......................................................................... 81
    8.5.1. Default Temporary Tablespace ............................................................ 81
    8.5.2. Space Allocation in a Temporary Tablespace ...................................... 82
    8.5.3. Creating a Locally Managed Temporary Tablespace ............................ 82
    8.5.4. Creating a Bigfile Temporary Tablespace ............................................ 82
    8.5.4. Multiple Temporary Tablespaces: Using Tablespace Groups ............... 83
  8.6. Creating a Tablespace Group ................................................................. 83
    8.6.1. Changing Members of a Tablespace Group ......................................... 83
  8.7. Specifying Nonstandard Block Sizes for Tablespaces .............................. 84
  8.8. Altering Tablespace Availability ............................................................. 84
    8.8.1. Taking Tablespaces Offline ................................................................. 84
    8.8.2. To rename or relocate tablespace datafiles ......................................... 84
    8.8.3. Bringing Tablespaces Online .............................................................. 85
    8.8.4. Using Read-Only Tablespaces ............................................................ 85
    8.8.5. Making a Tablespace Read-Only ........................................................ 86
    8.8.6. Making a Read-Only Tablespace Writable ......................................... 87
  8.7. Altering a Locally Managed Temporary Tablespace ................................ 87
    8.7.1. Shrinking a Locally Managed Temporary Tablespace ......................... 88
  8.8. Dropping Tablespaces ........................................................................... 89
10. Undo Management ................................................................. 106
  10.1. What Is Undo? ................................................................. 106
  10.2. Introduction to Automatic Undo Management ......................... 106
    10.2.1. Overview of Automatic Undo Management ............................. 106
  10.3. About the Undo Retention Period ........................................ 107
    10.3.1 Automatic Tuning of Undo Retention .................................... 107
    10.3.2. Retention Guarantee .......................................................... 108
    10.3.3. Undo Retention Tuning and Alert Thresholds .......................... 108
    10.3.4. Tracking the Tuned Undo Retention Period ............................ 108
    10.3.5. Setting the Minimum Undo Retention Period ........................... 109
  10.4. Sizing a Fixed-Size Undo Tablespace .................................. 109
  10.5. Creating an Undo Tablespace ............................................. 110
    10.5.1. Using CREATE DATABASE to Create an Undo Tablespace ........... 110
    10.5.2. Using the CREATE UNDO TABLESPACE Statement ....................... 110
  10.6. Altering an Undo Tablespace .............................................. 110
  10.7. Dropping an Undo Tablespace .............................................. 111
  10.8. Switching Undo Tablespaces ............................................... 111
  10.9. Establishing User Quotas for Undo Space ............................... 111
  10.10. Managing Space Threshold Alerts for theUndo Tablespace ......... 111
  10.11. Migrating to Automatic Undo Management ............................ 112
  10.12. Undo Space Data Dictionary Views ....................................... 112

11. Oracle-Managed Files (OMF) ................................................. 114
  11.1. What Are Oracle-Managed Files? ......................................... 114
  11.2. Who Can Use Oracle-Managed Files? ..................................... 114
  11.3. What Is a Logical Volume Manager? ...................................... 114
  11.4. What Is a File System? ..................................................... 115
  11.5. Benefits of Using Oracle-Managed Files ................................. 115
  11.6. Oracle-Managed Files and Existing Functionality ........................ 115
11.7. Enabling the Creation and Use of Oracle-Managed Files .......................................................... 115
  11.7.1. Setting the DB_CREATE_FILE_DEST Initialization Parameter ........................................ 116
  11.7.2. Setting the DB_RECOVERY_FILE_DEST Parameter ............................................................ 116
11.8. Creating Oracle-Managed Files .............................................................................................. 117
  11.8.1. How Oracle-Managed Files Are Named ............................................................................. 117
  11.8.2. Creating Oracle-Managed Files at Database Creation ................................................... 118
11.9. Creating Datafiles for Tablespaces Using Oracle-Managed Files .......................................... 122
11.10. Creating Control Files Using Oracle-Managed Files ......................................................... 124
11.11. Creating Redo Log Files Using Oracle-Managed Files .................................................... 125
11.12. Creating Archived Logs Using Oracle-Managed Files ..................................................... 125
11.13. Behavior of Oracle-Managed Files ........................................................................................ 126
11.14. Dropping Datafiles and Tempfiles ..................................................................................... 126
11.15. Dropping Redo Log Files ...................................................................................................... 126
11.16. Renaming Files .................................................................................................................. 127
11.17. Scenarios for Using Oracle-Managed Files ......................................................................... 127

12. Tables Management .................................................................................................................. 131
  12.1. About Tables ......................................................................................................................... 131
  12.2. Guidelines for Managing Tables .......................................................................................... 131
  12.3. Design Tables before Creating Them .................................................................................. 131
  12.4. Consider Parallelizing Table Creation ............................................................................. 132
  12.5. Consider Using NOLOGGING When Creating Tables .................................................... 132
  12.6. Consider Using Table Compression .................................................................................. 132
     12.6.1. Adding and Dropping Columns in Compressed Tables .............................................. 133
     12.6.2. Compression and Partitioned Tables ......................................................................... 133
     12.6.3. Determining If a Table is compressed .................................................................. 133
  12.7 Consider Encrypting Columns That Contain Sensitive Data ............................................... 134
  12.8. Restrictions to Consider When Creating Tables ................................................................ 134
  12.9. Creating Tables .................................................................................................................. 135
     12.9.1 Parallelizing Table Creation ...................................................................................... 137
  12.10. Automatically Collecting Statistics on Tables .................................................................... 139
  12.11. Altering Tables .................................................................................................................. 139
     12.11.1. Reasons for Using the ALTER TABLE Statement .................................................... 139
  12.12. Redefining Tables Online .................................................................................................. 143
     12.12.1 Features of Online Table Redefinition ..................................................................... 143
     12.12.2. Performing Online Redefinition with DBMS_REDEFINITION ............................... 144
     12.12.3 Creating Dependent Objects Automatically ........................................................ 146
     12.12.4. Creating Dependent Objects Manually .................................................................. 146
     12.12.5. Results of the Redefinition Process ....................................................................... 146
     12.12.6. Performing Intermediate Synchronization .......................................................... 147
     12.12.7. Aborting Online Table Redefinition and Cleaning Up After Errors ....................... 147
     12.12.8. Restrictions for Online Redefinition of Tables ....................................................... 147
  12.13. Online Table Redefinition Examples ............................................................................... 148
  12.14. Privileges Required for the DBMS_REDEFINITION Package ..................................... 153
13. Indexes Management ................................................................. 171

13.1. About Indexes................................................................. 171

13.2. Guidelines for Managing Indexes......................................... 171

13.2.1. Create Indexes after Inserting Table Data.......................... 171

13.2.2. Index the Correct Tables and Columns.............................. 171

13.2.3. Columns That Are Suitable for Indexing............................ 172

13.2.4. Order Index Columns for Performance.............................. 172

13.2.5. Limit the Number of Indexes for Each Table...................... 172

13.2.6. Drop Indexes That Are No Longer Required...................... 172

13.2.7. Estimate Index Size and Set Storage Parameters............... 172

13.2.8. Specify the Tablespace for Each Index............................ 173

13.2.9. Consider Parallelizing Index Creation............................. 173

13.2.10. Consider Creating Indexes with NOLOGGING.................... 173

13.2.11. Consider Costs and Benefits of Coalescing or Rebuilding Indexes 173
13.2.12. Consider Cost before Disabling or Dropping Constraints .......................................................... 174
13.3. Creating Indexes................................................................................................................................. 174
  13.3.1. Creating an Index Explicitly ........................................................................................................... 174
  13.3.2. Creating a Unique Index Explicitly ............................................................................................... 175
  13.3.3. Creating an Index Associated with a Constraint ........................................................................... 175
  13.3.4. Specifying Storage Options for an Index Associated with a Constraint ........................................ 175
  13.3.5. Specifying the Index Associated with a Constraint ........................................................................ 175
  13.3.6. Creating an Index Online ............................................................................................................... 176
  13.3.7. Creating a Function-Based Index ................................................................................................... 176
  13.3.8. Creating a Key-Compressed Index .................................................................................................. 177
  13.3.9. Creating an Invisible Index ............................................................................................................. 177
13.4. Altering Indexes ................................................................................................................................. 178
  13.4.1. Altering Storage Characteristics of an Index .................................................................................. 178
  13.4.2. Rebuilding an Existing Index ........................................................................................................ 178
  13.4.3. Making an Index Invisible ............................................................................................................. 179
  13.4.4. Monitoring Index Usage ................................................................................................................. 179
  13.4.5. Monitoring Space Use of Indexes ................................................................................................... 179
  13.4.6 Indexes Data Dictionary Views ....................................................................................................... 180
14. SQL*Loader ........................................................................................................................................ 181
  14.1. SQL*Loader Parameters .................................................................................................................... 181
  14.2. SQL*Loader Control File .................................................................................................................. 182
  14.3. Input Data and Datafiles .................................................................................................................. 182
    14.3.1. Fixed Record Format ..................................................................................................................... 182
    14.3.2. Variable Record Format ................................................................................................................. 183
  14.4. LOBFILEs and Secondary Datafiles (SDFs) ..................................................................................... 183
  14.5. Data Conversion and Datatype Specification ................................................................................... 184
  14.6. Discarded and Rejected Records ..................................................................................................... 184
    14.6.1. The Bad File ....................................................................................................................................... 184
    14.6.2. SQL*Loader Rejects ..................................................................................................................... 184
    14.6.3. Oracle Database Rejects .............................................................................................................. 184
    14.6.4. The Discard File ............................................................................................................................ 184
  14.7. Log File and Logging Information ................................................................................................... 185
    14.8.1. Conventional Path Loads .............................................................................................................. 185
    14.8.2. Direct Path Loads .......................................................................................................................... 185
    14.8.3. Parallel Direct Path ....................................................................................................................... 185